HP 9000 Series 300 Computers

HEWLETT PACKARD

NMODE User's Guide

LISP
LISP
LISP
LISP
LISP
LISP

NMODE User's Guide

for HP 9000 Series 300 Computers

HP Part Number 98678-90020

Hewlett-Packard Company
3404 East Harmony Road, Fort Collins, Colorado 80525

Printing History

New editions of this manual will incorporate all material updated since the previous edition. Update packages may be issued between editions and contain replacement and additional pages to be merged into the manual by the user. Each updated page will be indicated by a revision date at the bottom of the page. A vertical bar in the margin indicates the changes on each page. Note that pages which are rearranged due to changes on a previous page are not considered revised.

The manual printing date and part number indicate its current edition. The printing date changes when a new edition is printed. (Minor corrections and updates which are incorporated at reprint do not cause the date to change.) The manual part number changes when extensive technical changes are incorporated.

May 1986...Edition 1. This manual documents release 1.0 of the common LISP Development Environment for HP 9000 Series 300 Computers.

May 1986...Update. Adds chapters missing from Edition 1.

HEWLETT PACKARD

Manual Update

This Manual Update is for HP Part Number: 98678-90020
Title: NMODE User's Guide
Printing Date: 5/86

This manual update provides new information that was inadvertently omitted when the *NMODE User's Guide* was published. We apologize for the oversight.

This update contains two chapters that may be missing from your manual and a revised Table of Contents and Printing History.

- Chapter 8: Directories Facility
- Chapter 17: Customization
- Replacement Table of Contents
- Replacement Printing History

To update the *NMODE User's Guide*:

Insert the chapters into the manual at the appropriate locations.

Replace the Table of Contents with the one provided in this update.

Replace the Printing History with the one provided in this update.

No changes are needed to the on-line version of the manual.

Hewlett-Packard Company • 3404 East Harmony Road • Fort Collins, Colorado 80525

HP Part Number
98678-90800
Printed in U.S.A. 5/86

98678-90800

Table of Contents

Chapter 1
The Overall Picture

Introduction

This **manual** describes how to use the NMODE environment. This **chapter** describes the overall nature of NMODE. It contains a basic introduction to NMODE that is assumed in later chapters.

To help you quickly locate the desired information in this manual, each chapter and major section has the following organization:

- First, an introduction describes the operational characteristics of the topic at hand. The introduction describes the functionalities you are about to investigate. This way, you will immediately know what you can learn by further reading.
- Next, any prerequisites such as loading a module or setting up an environment are described. When many alternatives for setting things up are available, the manual mentions one or two efficient commands or procedures that let you get started. In addition, in cases where it is desirable to describe some fundamental concepts related to NMODE, this information is included.
- After any prerequisites are explained, the chapter or section discusses the various things you can accomplish, relative to the topic at hand, by explaining commands, routines, and procedures. The manual explains multiple ways to do things and often lets you decide how to proceed, depending on how you like to work and what you want to do.
- Finally, each chapter or major section discusses the various details related to the topic.

You do not need to use all of NMODE to work effectively, and correspondingly, you do not need to read the entire manual at one time. The manual is segmented and each segment is graduated so that you can obtain some fundamental information in a few chapters and then get information selectively, depending on what you want to do. Included with the system are on-line tutorials. You may wish to use both this manual and the tutorials as you learn about NMODE.

To see the total organization and content of this manual, you might take a moment to examine the table of contents, which is the major guide to finding information by topics. For specific items, the index references information by label and page number.

The information in this manual relates to the NMODE user environment as you received it. This point might seem trivial, but later, you will see that NMODE can be customized or extended, or both. If you customize or extend NMODE, your version might not work according to what is said in this manual. If you intend to alter your environment and you are concerned about being able to restore the original NMODE user environment, discuss with your appropriate Hewlett-Packard representative about the best way to backup the NMODE software you received.

Many parts of NMODE are like the EMACS editor originally developed by Richard M. Stallman at M.I.T. NMODE itself was based to some degree on EMODE, which was developed by William F. Galway at the University of Utah.

Before You Read Further

You need to do the following things before you read and use this manual.

- Assemble and check your hardware system.
- Install HP-UX 5.1 (or later version)
- Install the Common Lisp development environment software.
- Install NMODE. This occurs in conjunction with installing Common Lisp, but be aware that you can do some customizing.
- Invoke NMODE. That is, you boot HP-UX and then run the NMODE user environment.

The last point might seem trivial, but this manual assumes that NMODE has been invoked; it does not deal with how to configure the software and invoke it. (That is explained in the *Installation and Overview* manual.)

As a quick review, if HP-UX is up and the development environment software has been installed and you have logged on and see a familiar shell prompt (% or $), then you can invoke NMODE by typing:

```
    $LISP/bin/nmode [Return]
```

The invocation process takes one to two minutes depending on what options you are loading.

What Is NMODE?

This section briefly describes what NMODE is in a conceptual sense. It does not tell you how to do anything. In general, depending on your background and orientation to software, NMODE is any or all of the following things:

- NMODE is an integrated program development environment. It provides an easy-to-use human interface that can be manipulated by use of a pointing device (mouse), simple keystrokes or English-like commands.
- NMODE provides an Emacs-type editor for rapid manipulation of text.
- NMODE supports the interactive development of Common Lisp programs and provides simplified editing, compiling, and error correcting of C, Pascal, and FORTRAN programs.
- NMODE allows easy access to the underlying HP-UX operating system. You can execute HP-UX commands in a shell-buffer which is similar to an HP-UX shell but retains all text in an NMODE buffer.
- Besides being just an interface with an EMACS-type editor, NMODE lets you develop, interpret, compile, and debug Lisp source code without ever leaving the editing environment! In addition, it provides browsers that contain code and search indexes which efficiently find source code functions and strings that you want to edit.
- You can use NMODE to develop code in C, Pascal, and Fortran. NMODE provides assorted templates that help you write code and also provides a browser that contains compilation error indexes. These error indexes take you directly to errors produced during compilation of C, Pascal, or Fortran source code. In addition, you can directly move between editing and browsing modes, which permits making smooth transitions from editing to compiling to debugging and otherwise manipulating source code.
- The name NMODE can also be applied to the library that contains the editor and assorted tools.

Although NMODE is very flexible, you will discover that it is predictable and straightforward. The major things you can do include:

- Edit, interpret, compile, and debug source code
- Prepare reports and other documents
- Manage files in directories

- Get online information
- Interact with the underlying Common Lisp language
- Interact with the HP-UX operating system
- Run applications

Where Does NMODE Fit In?

The following general illustration shows the major parts of NMODE and the relationship of NMODE within the Common Lisp Development Environment. The following illustrates the high-level structure of NMODE.

```
                 Buffers (provides an Emacs Mode editor)

                 Directories (provides directory and file management)

                 Documentation (provides online documentation)

NMODE            Additional Facilities (provides tools for programmers)

                 User Options (provides control of user customization)

                 Code Indexes (provides indexes to source code)

                 Search Indexes (provides indexes to any text)

                 Compilation Error Indexes (provides indexes to errors)

                 User Developed Applications


Common Lisp


Optional software (e.g. Window Management)


HP-UX
```

Notice the following things:

- HP-UX is the underlying operating system. It provides a base for Common Lisp and other software. NMODE can have a shell-buffer which lets you execute HP-UX commands as if you were in a shell.

- You can run optional software on HP-UX that adds functionality; for example, window management.
- Common Lisp runs on HP-UX and is independent of NMODE. Thus, a programmer can use NMODE to develop Lisp-based applications which can run later without NMODE.
- NMODE runs on Common Lisp and interfaces to the underlying Common Lisp environment; for example, you can interpret or compile Lisp source code while you are in NMODE.

Overall, the illustration shows that HP-UX, Common Lisp, and NMODE provide a base for a software development environment. Other software can be added to extend the environment to provide the functionality you need. The illustration also implies that HP-UX and Common Lisp can provide a basic execution environment on which you can run assorted applications.

Finding Information

This manual, which describes the use of NMODE, is part of the manual set for the Common Lisp Development Environment. The set contains the following additional guides and manuals:

- The *Installation and Overview* manual describes how to install the HP-UX operating system, the Common Lisp environment, and the NMODE user environment. Working through the installation manual is a prerequisite to using this user's guide.
- The *NMODE Command Reference* manual supplements this user's guide in that it contains reference information about each NMODE command.
- The *Lisp Programming Guide* describes assorted programming topics. The guide explains how to use Common Lisp and assumes that you are familiar with Lisp.
- The *Lisp Language Reference* manual contains reference information about Common Lisp.
- The documentation set for HP-UX, especially the *HP-UX System Administrator* manual, provides a detailed explanation of the HP-UX operating system.

Getting Information in This Manual

The Table of Contents provides a quick means of seeing all of the topics discussed in this manual. The appendices provide the following types of reference information.

- *Appendix A* is a lift-out card that lists all typical NMODE Commands. For each command, the appendix provides: a descriptive phrase, a key sequence, and a Meta-X command name. The commands are grouped into topical categories so you can find them quickly .
- *Appendix B* is a lift-out card that contains an alphabetized list of special browser commands. The appendix lists command names and describes the commands.
- *Appendix C* is a lift-out card that lists and describes the commands you can execute by pressing a softkey. In most cases, pressing a softkey merely provides a convenient way to execute a certain command. In a few cases, pressing a softkey toggles or cycles certain states or sets of softkeys.
- *Appendix D* contains information about the keyboard. Examine this appendix if you want to get information about bindings for keys.

Although NMODE is an integrated environment, information is presented in this manual in somewhat separate pieces so that you can quickly find the information you want.

This manual does explain those commands that let you work with other software. For example, you will probably use window management software. Commands that manipulate windows are documented in this manual. Similarly, this manual documents mouse commands, but does not describe the HP 46060A device in general.

Looking at NMODE

NMODE is essentially a user environment for software development. For example, with NMODE you can: write source code, interpret and/or compile the code depending on the language. This section describes the main features of that environment.

Some Basic Definitions

Before reading other chapters, it is necessary to define some terms that are used throughout this manual to provide a consistent way to talk about the things you work with.

NMODE Root	The many parts that make up NMODE are not just a collection; they have hierarchical and integrated relationships. When you invoke NMODE, you get a screen-oriented interface that provides a list of facilities. This top level is named `NMODE Root` in the screen display that lists **facilities**. Do not confuse references to `NMODE Root` with any references to the *root* of the HP-UX file system or the *root* user (or system administrator).
Facility	When you are at NMODE Root, the display shows a list of items (such as `Buffers`, `Directories`, and `Documentation`). Each item that is listed at NMODE Root is called a **facility**, and the collection is called **the facilities**.
Item	Each facility contains a list of items. In using NMODE, you typically get into a certain facility and then work with its items. The generic term, **item**, is used to refer to any of a number of "things" you might be using. For example, the items in the `Buffers` facility are buffers (workspaces) in which you can edit text. Other facilities contain other types of items.

In working with NMODE, you will notice that NMODE provides a consistent interface for using the items. Two mechanisms, one called a browser and the other an EMACS-type editor, stand out. The mechanism called a **browser** lets you manipulate assorted facilities and items; for example, directories, files, indexes, online help, and user options. In a different vein, the second mechanism is an **EMACS-type editor** that lets you edit text in working buffers.

The Features

The following terms and information describe NMODE's features, and thereby indicate what you can do within NMODE.

- **Integrated**: Without going into details, this term means that NMODE's files work together to provide a total user environment. Here are three examples:
    - NMODE's structure and available commands make it easy for you to sequentially point to and browse a facility or item regardless of what it represents. For example, starting at the NMODE Root, you could successively point to and browse: the `Directories`, an active directory, and a particular file. Then, you could "switch gears" and edit the file. The term, **point to**, means that you move a highlighted region onto an item in a browser's list. The item that you are pointing to is also referred to as the current, selected, or active item.
    - NMODE is implemented entirely in Lisp. You can evaluate Lisp forms or compile and execute them, without leaving NMODE's editing environment!
    - The availability of multiple buffers and windows lets you edit several files in parallel; for example, source code, documentation, and report files. You can pause at any time, leave what you are doing, work with other items in the environment, and then return to your files.
    - You can access HP-UX in several ways without leaving NMODE.
- **Customizable**: You can change the key sequences that invoke commands or add certain features to the environment.
- **Extensible**: Beyond customization, you can write functions for new commands, bind them to key sequences, and add them to your environment. You can also replace parts of NMODE incrementally.
- **Screen-oriented**: Typed characters automatically echo to the appropriate region of the screen. Characters typed for text or code appear in a section of a window called a **pane**. Characters typed for commands or entries related to prompts appear in a **message area**. This echoing occurs interactively in the sense that the display is updated often, usually after entry of each character.
- **Documented**: NMODE provides documentation in online manuals and frames of information that can be accessed in various ways and displayed on the screen. Among other things, you can get information about commands, terms, and definitions. A later chapter of this manual called *Getting Online Help* contains detailed information about getting online help.

- **Moded**: The name, NMODE, derives from the fact that the user environment has multiple modes. Here is a general scheme for thinking about modes.
  - In NMODE, you are always using a certain type of item; for example, a facility (such as a directory), a file, or a buffer.
  - An item always has some associated mode; for example, a buffer has an editing mode.
  - The mode determines available commands. That is, the available commands are always suited to the items at hand.
  - A mode consists of one **major** mode and zero or more **minor** modes. The major mode provides the bulk of available commands while the minor mode provides a limited set of special commands, generally suited to a particular type of item. For example, in Emacs mode there are many commands for manipulating text. If we add the HP-UX minor mode, there are now additional commands specialized for working with HP-UX.

The current set of modes is based on ongoing development of user environments that began at M.I.T. in the 1960s with the development of TECO and continued with the development at M.I.T. of EMACS. The modes developed in these editors are the predecessors of the current modes in NMODE.

With these terms in mind, the following chart shows relationships among the entities that make up the NMODE user environment.

```
                    Basic Screen-oriented Interface
       (The "mechanism" for using the NMODE's items and modes)


                    The total NMODE user environment


                           The Facilities
               (Provide access to all items in NMODE)
      [Directories, Documentation, Buffers, User Options, Code, and Others]


                             The Items
             (Are the entities you primarily work with)
    [files, text, information, forms, indexes, system variables]
```

These things work as follows:

- The basic screen-oriented interface simply lets you work with the things in NMODE. This interface is fully described in the next chapter.
- The **facilities** let you access all parts of NMODE and sits at the top level, called NMODE Root. In general, you do not work with facilities so much as you use them to get to items they contain.
- The **items** are the "nuts and bolts" things that you manipulate as you browse files, edit text, search for code items, and so on.

With this in mind, we can now begin to use NMODE.

Using NMODE

Answering the question, "How does one use NMODE?" is not trivial since there is no single way to use any part of NMODE. Many of the common operations, such as editing or manipulating files, can be accomplished in several ways. While this may appear confusing to the beginner, you should not worry because you can quickly learn the fundamentals.

Remember that each chapter and section provides some initial description and quickly points out things you can do. Then, the chapter or section explains available alternatives and lets you decide how you want to operate.

Overall, the following information suggests how to use NMODE.

- NMODE provides a **screen-oriented interface** that has several regions. Each region serves a certain purpose. For example, the **mode line** always indicates the state of the current environment. The next chapter called *Basic Use of NMODE* explains this interface in detail.

- You can go to the part of the environment that you want and then complete tasks by executing commands that operate on current items in a current mode. In most cases, you invoke a particular browser that lets you manipulate items, or you invoke the editor to edit text.

- When you are in a browser, you can examine a list of items, point to an item, and then browse it. For example, starting at the `NMODE Root`, you could point to and browse `Documentation` to get a list of online manuals and help text. Then, you could point to and browse an online manual to get a list of chapters. In addition to pointing and browsing, you can execute any of several commands that let you manipulate the items.

  When you are in a browser, the current environment provides a menu of special browser commands that let you do such things as browse, filter, or group items. The browser commands, which appear at the bottom of the screen, are always tailored to the current items and mode and allow a menu driven routine for manipulating items.

  When you are in a text buffer, no special menu of commands is available (unless you use the mouse to get a popup menu), but you can write and edit text, evaluate source code, and so on. Many commands are available for editing. These commands are EMACS like commands. Appendix A contains a quick reference to them.

- NMODE has various commands that let you move the cursor, manipulate files, compile source code, manipulate windows, and otherwise manage the overall environment. Here are two examples:

The editing modes provide commands for getting a file to edit, deleting items such as words, lines, sentences, paragraphs, and regions. Having deleted an item, you can move the cursor to a desired location and reinsert the deleted item. You can then save the edited text in a file.

After you write Lisp source code and edit it to your satisfaction, you can immediately interpret the code to see if it works. If it does work, you can save the code to a file and immediately compile the file.

When learning to use NMODE, most people begin by using a point and browse method to perform tasks. Some use menu driven methods to execute certain browser commands. Later, most people learn several commands that provide direct access to items they wish to use.

The Next Step

This chapter, which you have almost finished, generally explained or implied what you can do without saying anything about how to do it. This means that the chapter implied the need to read other chapters. You will eventually need to work through subsequent chapters if you want to make use of all parts of the NMODE environment. Here is what they contain.

- *Basic Use of NMODE* is essential reading. You cannot effectively use NMODE unless you are acquainted with the screen-oriented user interface, the types of modes that provide commands, the ways to execute commands, and various related details. The chapter contains much information, but you will soon learn how to manage and use it.
- The chapter called *Getting Online Help* explains how to have the system provide information when you need it.
- The chapter named *The HP-UX Facility* explains the facility that lets you interact with the underlying HP-UX operating system without having to leave NMODE.
- Each browser is discussed in a separate chapter that you can read **after** you read the chapter called *Introduction to Browsers*, which explains browsers in general.
- *Working With Text* contains information related to editing human-readable text (as opposed to program source code). This information is assumed in later chapters called *Working With Lisp Code* and *Working with Non-Lisp Code.*
- *Working With Lisp Code* and *Working with Non-Lisp Code* contain information related specifically to editing and otherwise manipulating source code.

- Customization is treated last in a separate chapter, should you wish to alter the user environment you originally received.

Be sure to read the chapters that contain introductory material before tackling the advanced topics. It is suggested that you take time to read the information and examine your system at the same time. Be sure to watch for and read about the **mode line**. This is important because, the mode line is always your best indicator of where you are located within the environment.

While few specific exercises are provided in any of the chapters, many examples are included. Try them out to see user-level cause and effect relationships. In particular, it is suggested that you work through the parts of chapters or sections that discuss prerequisites to doing something so you can see first-hand what the system does when you load a module or set values for certain user options. Then, read additional material selectively and perform tasks related to what you need to accomplish.

Chapter 2
Basic Use of NMODE

Introduction

This chapter describes how to use the NMODE environment. Here you can learn how to interpret the display, execute commands, and use assorted routines for moving around in the environment. In discussing these types of operations, specific information about conventions and procedures is included so that, as you read later chapters, you will know how to interpret particular situations. By knowing this type of information and by knowing about NMODE's "parts", you will soon be able to use the environment in very powerful ways.

Specifically, by reading the chapter you can learn how to:

- Understand and use the screen-oriented interface. Following this, the basic structure of NMODE is illustrated and discussed.
- Use the keyboard, softkeys, and pointing device (mouse) to execute NMODE commands.
- Work with NMODE's modes. You will see how the current mode provides the commands that let you work productively with current items.
- Use several different routines to access and otherwise work with items.
- Supply numeric arguments to a command being executed.
- Cancel, abort, or exit commands or situations.

Each of these things will be useful as you read subsequent chapters.

The Display

This section explains what you see on the terminal or window(s) that NMODE is running on. The information is important because NMODE's interface is **screen-oriented**; it lets you know what the system is doing by displaying items, text, prompts, and menus in certain areas. This "partitioning" of the display is the fundamental principle underlying the interface. For example, by reading further, you will see how the **mode line** always lets you determine where you are within the overall environment.

While NMODE can be used with or without window-management software, having window-management software provides various conveniences such as multiple NMODE windows and popup menus. Exceptions to NMODE's behavior when it is running without the window manager will be noted.

The NMODE Display

An **NMODE window** is the fundamental interface to NMODE. Without window-management software, NMODE uses the entire physical display as its window. With window-management software (Windows/9000), NMODE windows correspond to Windows/9000 windows.

For simplicity, the term **NMODE window** will refer to the entire video display when running without window management software, and it will refer to the one or more Windows/9000 windows used by NMODE when you are using window management software. (While it is possible for NMODE to use more than one video display, such configurations are not discussed in this chapter.)

The NMODE screen is partitioned into panes, message areas, and mode lines. Such things as text, lists, names (titles), messages, menus, softkey labels, prompts, user entries (input), and system related information appear in certain locations, according to the their purposes.

For the sake of simplicity, only one NMODE window appears in the following illustration and description of the screen display.

The illustration contains illustrative entries in certain areas to demonstrate how the areas are used. In the illustration, the display is enclosed, and the labels for areas appear on the left side.

```
| ------------------------------------------------------------------------
|               | ----------------------------------------------------  |
|               | <area for browser list or buffer contents>          | |
|               |                                                     | |
|               P                                                     | |
|               A                                                     | |
|               N                                                     | |
|               E                                                     | |
|               |                                                     | |
|               |                                                     | |
|               |                                                     | |
S               |                                                     W |
C  Mode line--> | I/O @ Emacs (Text Fill) /users/julie/stuff.text -04%- *  I |
R               | ____________________________________________________N |
E               | <another area for browser list or buffer contents>  D |
E               |                                                     O |
N               P                                                     W |
|               A                                                     | |
|               N                                                     | |
|               E                                                     | |
|               |                                                     | |
|  Mode line--> |      Emacs (Lisp) [OUTPUT] *                  12 LISP| |
| Message area->| C-M-L to QUIT,  C-? for HELP, C-X R for Root        | |
|               | ____________________________________________________| |
|                                                                       |
| Softkey        Help  Next   Nmode   Nmode  Buffers  Direc-  Options   HP-UX |
|   labels -->         Keys  Command  Root            tories          Command|
| _______________________________________________________________________|
```

The General Screen Display

This is the general layout of a screen containing an NMODE window. The screen can be more complex, depending on your system and what you are doing. To help you get an overall notion about what appears on the screen and how it works, the following subsections describe each area in the NMODE window and the important items that appear in those areas.

Physical Display

The entire physical display is one screen. From the bottom to the top, it contains softkey labels, and a window. With Windows/9000 running, the screen can have multiple NMODE and HP-UX windows, and you can move freely among them.

Window

An NMODE window sits inside the screen and incorporates one or more panes and a two-line message area. With window-management software, a window has a name and an optional border. The border has regions, especially in the corners, that provide certain window management functionalities. The diagonal arrow lets you move the window; the square lets you make the window into an icon, the square with the smaller square lets you make the window smaller.

Pane

Panes sit inside a window. The illustration shows two panes. Each pane has a highlighted mode line and contains either **text** (if the pane is displaying a buffer) or **items** (if the pane is displaying a browser).

Most of the time, you focus on the contents of a pane as you use NMODE and note the contents of other areas to get supplemental information; for example, the name of an associated file or a browser command. The pane for the OUTPUT buffer is special in that its mode line also contains information related to the state of the underlying Common Lisp system.

Mode Line

Each pane contains a mode line. This line is very important since it tells you many things about the state of NMODE. In the previous illustration, the following two mode lines were shown:

```
I/O @ Emacs (Text Fill) /users/julie/my-stuff.txt -04%- *
```

and

```
            Emacs (Lisp) [OUTPUT] *                  12 LISP
```

In general, a **mode line** can contain some or all of the following information fields.

```
I/O @ Major  (Minors)  [BUFFER]  file  {PACKAGE}   -position-  *
```

When you look at a mode line for a pane, here is what each part tells you:

I/O

The I/O indicator appears when NMODE is idle for a certain amount of time and is waiting for input or output. A PWR indicator appears during the "power-up" of NMODE. A blank in this spot indicates that the system is busy executing. This indicator appears only in the current pane.

The "at sign" is an indicator for the current pane (i.e. the pane that is listening to the keyboard). You can change the indicator to another string by using the NMODE General Options Browser.

Major	This is a string that names the major mode; for example, `Emacs` for editing text or `Browser` for manipulating listed items. Only one major mode is current at a particular time.
(Minors)	The string or strings inside the parentheses name one or more minor modes; for example, `NMODE Root`, `Text`, `Lisp`, or `Directories`. A minor mode can be global, local, or tied to a major mode, depending on the implementation of the mode.
[BUFFER]	The string inside the brackets names the editing buffer when it differs from an associated file name. Otherwise, no buffer name is displayed. While only one buffer is current at one time, you can have many buffers and can move freely among them.
file	This is the absolute pathname for the associated file for the current editing buffer; for example, `/users/laurie/test22/codex.l`. The displayed pathname is for the last file visited in the buffer. You can visit any sequence of files in an editing buffer, reading the mode line to keep track of which file is current.
{PACKAGE}	When it is present, the string inside the braces names the current package; for example, `{LISP}`, `{USER}`, or `{NMODE}`. If the string is not present, you can execute **M-X show package** to see the current package.
-position-	The position is a percentage of the contents of the buffer that lies above the cursor (point); for example, `-84%-`. The position indicates `TOP` or `BOT` when the percentage approaches 0 or 100, respectively.
*	An asterisk appears when the contents of the buffer have been modified since the last save. Otherwise, the asterisk is not displayed.

Here are some examples of typical mode lines.

- This mode line is for the browser that contains the facility named `Buffers`. The major mode is `Browser` and the minor mode is `Buffers`.

  ```
  Browser (Buffers)
  ```

  The pane is not current (no leading string), and the mode line only indicates the major and minor modes.

- This mode line is for a current editing buffer in Emacs major mode with Lisp minor mode:

```
@ Emacs  (Lisp)  [MESS]  {USER}
```

The @ indicates that the pane is current. The buffer name is MESS. It has no associated file and has not been modified (no asterisk). The current package is USER.

- This mode line is for a current editing buffer in Emacs major mode with Text and Auto Fill minor modes:

  ```
  @ Emacs (Text Fill)  /users/john/info/ace-doc-23.txt --34%-- *
  ```

  No buffer name is displayed, but the buffer has an associated file which is indicated by the pathname, /users/john/info/ace-doc-23.txt. The -34%- indicates you are 34% of the way through the buffer. The asterisk indicates the buffer has been modified.

  By the way, you could invoke the browser for Buffers to verify that the buffer name is ACE-DOC-23.TXT. Whenever you execute a command that finds a file, a buffer is created for the file and has the file's name even though the buffer name is not displayed in the mode line.

Regarding the mode line, NMODE has one special case. The mode line for the OUTPUT buffer displays typical information and also displays information about the current state of the underlying Lisp system. The right end of the OUTPUT buffer mode line looks something like this:

```
9 LISP
```

The 9 indicates the number of calls to the Lisp interpreter during the current NMODE session. This number increases by one each time you evaluate a Lisp form.

When evaluation of a Lisp form produces an error, the mode line for the OUTPUT buffer displays information about break loops; for example:

```
10 DEBUG (3)
```

The 10 indicates that you have called the Lisp Interpreter 10 times. The DEBUG indicates you have entered the debugger. The (3) indicates the number of levels down from the top level of the current break loop.

Everything you see in a pane is in a current buffer and that buffer has a current major mode and some number of minor modes. The current modes provide the functionalities for the buffer; they control for example, whether the contents of the buffer are items you can manipulate or text you can edit. Information about the current mode appears in the mode line for the current pane. In general, the mode line shows the current state of NMODE.

Message Area

Notice that the message area in the illustration contains a reminder:

```
C-M-L to QUIT,  C-? for HELP, C-X R for Root
```

that tells you how to exit the buffer, get help, or return to the top level of NMODE. The message area is used for

- Prompts for commands. Many commands are interactive, prompting for user input. Commands that prompt for user input temporarily invoke an Input mode that lets you edit the input with normal editing commands, get help, or abort the command. Input mode might also provide additional functionalities (i.e. pressing the space bar or [ESC] to complete a filename or pressing [Enter] or [Return] to terminate input).
- Menus for browser commands (also called a command line).
- System and error messages. For example, if possible, the system indicates when it is collecting garbage. The information returned by many commands appears in the message area. **C-=** is such a command.
- Strings (user input) typed as part of executing certain commands appear under the prompt.
- Results returned by using the Evaluate Form command, **C-X C-E**, to evaluate Lisp forms.

The message area is part of a window. A multiple window system contains a message area for each window. During sessions with NMODE, your attention is often directed to the message area as you interact with the system or get information. Some messages appear only briefly, so you need look for them before they are erased. Like the mode line, the message area often tells you what the system is doing.

Softkey Labels

The softkey labels appear at the bottom of the screen. You might need to press [Menu] to display the labels. In general, the softkey labels change to fit the current mode and state of the NMODE environment.

Selected Window Name

If you have window-management software, and the softkey labels are being displayed, the lower-right corner of the screen displays the name of the currently selected window. Below this, NMODE will indicate when a garbage collect occurs by printing "*** GC ***".

Cursor
The currently selected pane contains a cursor. The cursor can be a blinking underscore beneath the current character or a rectangular region on the current character, which shows through in inverse video. In a browser, the cursor normally sits at the left end of the highlighted region, but it can be moved to a character or word in the line.

In a buffer, the cursor tracks the "point", which is a position (column and row) in the buffer just to the left of the cursor where characters are entered. When you execute certain commands that invoke a temporary Input mode, the cursor moves to the message area but it will return to the location of the point within a buffer.

Besides using a point to keep track of the location for character-entry in an editing buffer, NMODE can use up to 16 "marks" to keep track of certain locations in the buffer.

Highlight
When you are in a browser, the current item is highlighted by a rectangular region that adjusts automatically to "fit" the item. The cursor sits somewhere in the highlight, usually at the left end.

Pointer
If you have window-management software and a locator device (mouse), a small pointer will appear on the screen. When you are inside a window, or in the softkey label area, the pointer appears as an arrow. If you are "outside" all windows, in the "desktop" areas of the screen, the pointer appears as a small square. The pointer changes to a "cross-hair" when you point to a border region or an icon.

Icons
An icon is a graphic figure which represents a window that is not currently displayed. With Windows/9000, any window can be hidden and replaced with an icon. Icons consist of a figure on top of a bar containing the name of the window it represents. The icon can be moved by placing the cross-hair of the pointer on the left end of the bar and pressing the left button. The icon can be "restored" to the window it represents by placing the cross-hair on the right end of the bar and pressing the left button on the mouse. Clicking the left button on the figure or the name invokes a popup menu for the window represented by the icon.

Popup Menus

NMODE supports "popup" menus when window-management software is running. The position of the pointer and whether the left or right button is pressed determines which popup menu will appear.

When the pointer is in the "desktop" area of the display, pressing the left button brings up the window-manager popup menu for the currently selected window. Pressing the right button brings up the **NMODE General** popup menu.

There are two slightly different cases when the pointer is inside an NMODE window.

- If the pane contains a browser, pressing the left button positions the highlight on the item nearest the pointer. If the pointer is already on the highlighted item, the item will be browsed. Pressing the right button produces a mode-sensitive popup menu that duplicates the command line menu.
- If a buffer is being displayed, the left button moves the cursor to the location of the pointer, while the right button provides a mode-sensitive popup menu. Pressing the left button when the pointer is at the current cursor position will set the mark at that point.

When the pointer is in the message area at the bottom of the window, you may select any command displayed there by pressing the left button.

To exit a popup menu, quickly move the pointer out of the menu, or press the space bar.

The Initial Screen Display

You previously examined the general screen display and read about the "things" it can contain. The following illustration shows the screen display when you invoke NMODE if your initialization files load the typical facilities.

```
NMODE Root

  Buffers
  Directories
  Documentation
  User Options
  Additional Facilities



    @ Browser (NMODE Root)

Help   Browse   Group   Filter  Create   Options   Kill   Quit
```

The Initial NMODE Screen Display

Notice that **NMODE Root** provides the facilities that let you access the files, buffers, and other items that you use to do productive work. NMODE has a hierarchical design allowing easy access to all parts of the system.

Note the following things about the previous illustration:

Title	The title (name) for the top level is `NMODE Root`. The title, which always appears when you are in a browser, is a good indicator of where you are located within the environment. A title is not present in an editing buffer.
Facilities	The root displays some of NMODE's facilities. These facilities contain the browsers that you use during working sessions.
Mode Line	The mode line is an important indicator of the current state of the environment. It was discussed previously in this chapter.
Message Area	Among other things, the message area displays the menu of browser commands that you can execute from the top level browser. Commands, prompts, messages, and inputs can appear in the message area.

From this top level, you typically use one of these methods to initiate a work-session with NMODE. The methods are explained later in detail.

- Point to a certain facility and browse into it. In general, the "point and browse" method is used to eventually get to a specific item under one of the facilities.
- Execute an NMODE command that takes you directly to a certain item or items. In general, the "direct-access" routine is used to circumvent sequential pointing and browsing when you know the name of the item you want to use.

You will see later that, besides knowing about the "entities" in NMODE, knowing how and when to use each method will help you work effectively and efficiently.

The Overall Structure of NMODE

The following illustration shows the generic structure of NMODE from the root to the second level, if you loaded several of the more commonly used facilities.

```
               BUFFERS
                    HP-UX.shell
                    MAIN
                    OUTPUT
                    <others>

               DIRECTORIES
                    $HOME
                    /
                    <others>

               DOCUMENTATION
                    Nmode User's Guide
                    Nmode Glossary
                    <others>

               ADDITIONAL FACILITIES
                    Execution Monitor
                    Execution Stack Analyzer
                    Code Index
                    Compilation Error Index
                    File Search Index
                    Program Editing Support
      HP-UX Access

               USER OPTIONS
                    Common Lisp User Options
                    NMODE Window Creation Options
                    HP-UX Access Options
                    NMODE General User Options
      Directory Options
                    <others>
```

With the exception of the Execution Monitor and Execution Stack Analyzer, each entity is explained somewhere in this manual; for those two, see the *Lisp Programmer's Guide.*

Using the Keyboard

This section explains the relationships among an NMODE **command**, the **name** of a command, the ways to use the keyboard to **execute** a command, and the **Lisp function** that implements a command. The section also describes how each key operates. Although a pointing device (mouse) is not part of a keyboard, the use of this device to execute commands is also described.

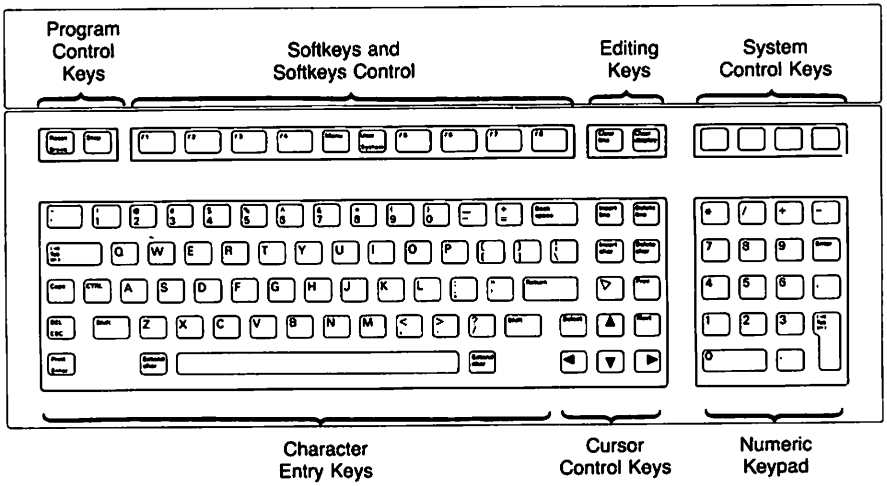
At the highest level of abstraction, your system utilizes a **virtual keyboard**, which is then mapped to a **physical keyboard**. In general, a function can be assigned to every key on the keyboard (this is called a **key binding**).

Some keys (e.g. Control and Meta) are not bound, to a specific function that implements a command. Instead, these keys modify the functions of other keys. That is, these keys are pressed with one or more keys to execute a command.

Appendix D, "Keyboard Mappings" describes the keys in the virtual keyboard. Refer to the appendix if you need detailed information.

The Physical Keyboard

Your system probably has an HP 46020A or HP 46021A keyboard. The following illustration shows such a keyboard.



The Keyboard

Guidelines for Using the Keyboard

The following explains the keys and what they do.

- The keyboard has the usual **QWERTY** arrangement of alphabetic, numeric, and special characters. Pressing these keys typically lets you input characters to buffers. When they are used with the Control or Meta keys, or with certain prefixes, these keys let you execute commands.

- On the right end of the keyboard is a **numeric keypad** whose keys generally duplicate keys in the main section of the keyboard,

- The keyboard has **softkeys** along the top row in two sets, which are a darker shade and are labeled $\boxed{\text{f1}}$ through $\boxed{\text{f4}}$ and $\boxed{\text{f5}}$ through $\boxed{\text{f8}}$.

- Most editing and cursor control keys work as you would expect. This includes $\boxed{\text{Insert line}}$, $\boxed{\text{Delete line}}$, $\boxed{\text{Delete char}}$, $\boxed{\text{Prev}}$, $\boxed{\text{Next}}$, and cursor movement keys ($\boxed{\blacktriangle}$, $\boxed{\blacktriangledown}$, $\boxed{\blacktriangleright}$, and $\boxed{\blacktriangleleft}$), The specific bindings for these keys are discussed later.

The next section discusses the specific key bindings for these keys.

Key Bindings

The term **key bindings** is used to describe the relationship between the key you press and the NMODE command that is invoked when you press the key.

Printing Characters

The typical **printing characters** typically insert themselves in an editing buffer. They do this because they are bound to the function called `insert-self-command`. The printing characters include: **A**, **B**, ..., **Z**; **a**, **b**, ..., **z**; and special characters such as **%**, **&**, and **@**.

Numeric Characters

The numeric characters (**0**, **1**, **2**, ..., **9**) are bound to the function named `argument-or-insert-command`. This means they either insert themselves as a character in a buffer or prompt, or they serve as a prefixed argument to a command (when preceded by **C-U**).

Special Key Operations and Bindings
The HP 46020A keyboard has **special keys** that let you execute commands; that is, the special keys are bound to functions that implement certain NMODE commands. For example, pressing Prev executes the Scroll Window Down Page command. Some special keys have "shifted" bindings that usually execute an "opposite" command. For example, pressing ⇱ moves the cursor to the beginning or a buffer while holding Shift and pressing ⇱ moves the cursor to the end of a buffer. In some cases, the shifted key can augment a command. For example, pressing ◄ moves the cursor one character to the left while holding Shift and pressing ◄ moves the cursor one word to the left. How this works, of course, is that pressing ◄ executes the Move Backward Character command, and executing Shift ◄ executes the Move Backward Word command. There is more explanation of this later.

Special Keys

The special keys do the following operations and have the following command bindings:

Meta	The left Extend char key is the **Meta key**. The Meta key is an integral part of NMODE, being similar to the Control key in that it is used to "modify" another key in a key sequence. This key is used by holding it down while you press another key. When you see this manual refer to something like **M-X <string>** or **M-<char>**. The **M** represents the Meta key and means that you should hold down the left Extend char. The sequence M-X is pronounced "Meta X"
CTRL	This is the Control key, which is often used in a key sequence to execute commands. This key is used by holding it down while you press another key.
Back space	This key moves the cursor backward one space in Emacs mode, deleting the existing character, and is bound to the Delete Backward Hacking Tabs command. In a browser, the key moves the highlight up one line. The shifted-key labeled DEL does almost the same thing as Back space and is bound to the Delete Backward Character command.
The Space Bar	In Emacs mode, this key inserts a space at the cursor's location. This key is bound to the Auto Fill Space command in Emacs mode with Auto Fill minor mode. In a browser, the key moves the highlight down by executing the Browse Next Item command. When pressed in input mode (in response to a prompt), it tries to complete the name of the command or file.

Caps	This key toggles the case for alpha characters.
Clear display	This key performs a full refresh of the NMODE screen and is bound to the NMODE Full Refresh command. A full refresh redisplays the screen, clearing the screen of any characters that should not be currently displayed.
Clear line	This key deletes the entire current line regardless of the location of the cursor and is bound to the Clear Line command. Unlike Delete line, the "cleared" line remains open so you can enter new text.
Delete char	This key deletes the character under the cursor and is bound to the Delete Character command.
Delete line	This key deletes the entire current line regardless of the position of the cursor and deletes the created open line. The key is bound to the Delete Line command.
Enter	This key, at the left of the keyboard, is bound to the Execute Form command in Lisp mode. It is bound to the Indent New Line command in Text mode. On prompts, it is bound to the NMODE Terminate Input command. Therefore, it executes a form, indents a line, or terminates input to a command, depending on the current mode. Its counterpart on the numeric keypad has the same bindings.
ESC	This key is not generally used by NMODE, but it is used by the Incremental Search command, **C-S**, and the Reverse Search command, **C-R**, to terminate a search when you have typed enough to find a desired string. Also tries to complete command or file names in input mode when you are responding to a prompt.
Insert char	When you are in an editing mode and pressed Delete char to delete a character, pressing Insert char reinserts the deleted character. If you delete several successive characters, pressing Insert char an equal number of times reinserts the characters in a reverse order, which restores the original characters. Otherwise, the key "beeps". The key is bound to the Insert Character command.
Insert line	This key opens a blank line below the current line and drops the cursor down, into the open line at the same relative location as it had. No text is pulled down onto the newline. The key is bound to the Open Line Indent command.

Menu	This key toggles display of the softkey labels on or off and is bound to the Display Menu Keys command when you have invoked window-management software.
Next	This key scrolls the display up one paneful and is bound to the Scroll Window Up Page command. The shifted Next selects the next screen and is bound to the Select Next Screen command. This difference is important because one command relates to the text in the current window (pane) and the shifted command relates to a different window in NMODE.
Prev	This key scrolls the display down one paneful and is bound to the Scroll Window Down Page command. The shifted Prev selects the previous screen and is bound to the Select Previous Screen command. This difference is important because one command relates to the text in the current window (pane) and the shifted command relates to a different window in NMODE.
Print	This shifted key lets you print the contents of the current buffer and is bound to the Print Buffer command.
Return	In a buffer Lisp mode, this key returns and indents appropriately and is bound to the Return And Indent command. In a buffer in Text mode with Auto Fill mode, Return effects a return (carriage return and line feed) and is bound to the Auto Return command; without Auto Fill mode, it is bound to the Return command. In a browser, Return executes the Browse Next Item command. In a prompt, pressing Return terminates input to a command and is bound to the NMODE Terminate Input command.
Shift	Holding this key down while you press another key accesses the binding for the top label on the key. For example, Shift User accesses the binding for User instead of the binding for System.
System	This key displays a certain set of softkey labels and is bound to the Display System Keys command when you have invoked window-management software. Pressing the softkey for a corresponding softkey label executes a command implied by the label.
User	This shifted key displays a certain set of softkey labels and is bound to the Display User Keys command when you have invoked window-management software. Pressing the softkey for a corresponding softkey label executes a command implied by the label.

◤	This key, which looks like a slanted triangle, moves the cursor to the beginning of a buffer and is bound to the Move To Buffer Start command. In contrast, $\boxed{\text{Shift}}$ ◤ moves the cursor to the end of a buffer and is bound to the Move To Buffer End command.
▲	This key moves the cursor or highlight up one line and is bound to the Move Up command. The shifted Up-arrow key scrolls the pane up one line, but does not move the relative location of the cursor, and is bound to the Scroll Window Up Line command.
▼	This key moves the cursor or highlight down one line and is bound to the Move Down command. A shifted Down arrow key moves the entire pane down one line, but does not move the relative location of the cursor, and is bound to the Scroll Window Down Line command.
◀	This key moves the cursor one location to the left and is bound to the Move Backwards Character command. The shifted Left-arrow key moves the cursor one word to the left and is bound to the Move Backward Word command.
▶	This key moves the cursor one location to the right and is bound to the Move Forward Character command. The shifted Right-arrow key moves forward one word and is bound to the Move Forward Word command.
$\boxed{\text{Break}}$	This key interrupts the Lisp process and causes the system to enter the debugger.
$\boxed{\text{Select}}$	This key is similar to the left button on the mouse.
$\boxed{\text{Tab}}$	In text mode, the tab key performs the standard tab function. In Lisp mode, this key performs the Lisp indenting function.

The $\boxed{\text{Reset}}$, and $\boxed{\text{Stop}}$ keys have no assigned functionalities.

The four **unmarked keys** at the top-right corner of the keyboard have no assigned functionalities.

The keys on the numeric keypad have bindings similar to their counterparts found on the keyboard.

Supplemental Key Bindings

Besides the above bindings for special keys, certain special keys also have bindings with the Meta key or Control-Meta keys (a later section explains how to execute these commands).

M-Backspace	This key sequence deletes the word preceding the cursor and is bound to the Kill Backward Word command.
M-Next	This key sequence selects the next screen and is bound to the Select Next Screen command.
M-Prev	This key sequence selects the previous screen and is bound to the Select Previous Screen command.
M-Return	This key sequence moves the cursor back to the previous indentation and is bound to the Back To Indentation command in Text mode.
C-M-Return	This key sequence, which uses the Control-Meta prefix, also moves the cursor back to the previous indentation and is bound to the Back To Indentation command in Text mode.

Executing NMODE Commands

This section explains the types of NMODE commands and the assorted ways to **execute** a command.

The sections focus on key sequences, M-X commands, and browser menus. Later, softkeys and other alternatives for executing commands receive some attention.

You can execute NMODE commands interactively from the keyboard, **and** you can execute NMODE commands noninteractively within Lisp code. This manual does **not** deal with writing code that executes NMODE commands noninteractively within code. The *NMODE Command Reference* manual mentions the Lisp functions that execute the interactive NMODE commands and mentions related noninteractive functions in some cases.

The general methods for executing commands via softkeys or a mouse are described later in the sections called "Softkey Commands" and "Locator Device Commands". All this will clear up as you execute commands and become familiar with NMODE. This is the general model. The next several sections describe specific ways to execute commands.

A Model for Executing Commands

This section is optional reading. It explains the relationships among the various concepts embodied in the idea of an NMODE command and how those concepts are used in this manual.

To help you get information and keep the idea of a command in perspective, this manual uses the following terms in a consistent manner.

- The **name** of a command is the basic identifer of a command. For example, the NMODE command that reads a file into a buffer (and does some related things) is called **the Find File command**. The name of a command is the basic reference for two reasons:
  - The *NMODE Command Reference* manual lists commands by name.
  - Commands can usually be executed by typing **M-X** followed by the command name.
- Each NMODE command is implemented by a **command function** written in Lisp. For example, the Find File command is implemented by the function `find-file-command`. The names of most NMODE command functions end in the word `command`.

- Each NMODE command can usually be **executed** in several ways, depending on such things as the current mode and your hardware/software installation. You typically have the following alternatives for executing a command.
  - A **key sequence** lets you execute a command by holding or pressing a series of keys. Throughout the manual, the key sequence for a command is usually mentioned with the name of a command. This helps you establish the connection between the name, which is used for reference, and the key sequence, which is used for execution. Key sequences appear in bold type so you can see them quickly. For example, **C-X C-F** is the key sequence for the Find File command.
  - A **M-X command** lets you execute a command by typing the Meta-X prefix, typing the name of a command, and pressing Return. A later section explains how this works. It is practical to use this alternative when no key sequence is available or when you know the name of a command and cannot remember any other means of executing the command. For example, **M-X find file**, executes the Find File command just as effectively as **C-X C-F**.
  - In many cases, pressing a special key on the keyboard executes a certain NMODE command. Of course, it is always possible to duplicate the function of a special key with a M-X command.
  - When you are in a browser mode, the system provides a menu which appears in the message area. You can execute these special commands by pressing the the key (case is insignificant) corresponding to the capitalized letter in the command's name.

Key Sequence Commands

A **key sequence** is a sequence of keys where the purpose of the initial key, or set of keys, is to provide a **modifier** that modifies the action of a final key. For example; holding down CTRL while you type F is a key sequence. The Control key is the modifier and the **F** is the final key.

The NMODE environment provides considerable flexibility in how you bind key sequences to commands. The "Customization" chapter in this manual has additional information on the Control and Meta keys. It also discusses related functions, global variables, and procedures for customization.

The key labeled Extend char at the left end of the keyboard is the Meta key in the NMODE environment. The key labeled CTRL near the left side of the keyboard is the Control key. This manual uses the following notations to describe key sequences that execute commands.

C-<char>	Hold down the Control key and press **<char>**.
C-X <char>	Hold down the Control key and press X, release them and press **<char>**.
C-X C-<char>	Hold down the Control key and press X, release, and then hold down the Control key and press **<char>**.
M-<char>	Hold down the Meta key (the left Extend char), and press **<char>**.
C-M-<char>	Some commands use the combination prefix, Control-Meta, in which you hold down the Control key **and** the Meta key **and** press **<char>**. Alternatively, you can execute the control-meta prefix with **C-Z**, and then press **<char>**.
Lisp <char>	This is the Lisp prefix, which means execute **C-]** and then press **<char>**.

NMODE has other prefixes. For example, **M-H** is the HP-UX interface prefix.

M-X Commands

Not every NMODE command can be executed via a key sequence, mouse, or softkey. But nearly every NMODE command can be executed via an M-X command. The term **M-X command** means that a command has a long invocation, which consists of the **M-X** prefix and a command name (English words that name the command). Presumably, the command name suggest the command's functionality, making it easier to remember. The term **M-X command** in this manual refers to a way to execute a command, not to the command itself.

Conventions for M-X Commands

The general convention for using M-X to execute a command is:

```
M-X  <string>
```

where string is the name of a command; for example, `find file`.

Let's examine this further. Executing **M-X** enters an input mode, which looks like this:

```
Enter command name (? for choices,  C-G to abort):
```

Enter a string such as `find file` or `nmode root` that is the name of a command and press Return. This initiates execution of the command.

If you execute **M-X** and then at some point in typing the command name forget the exact name of the command, typing `?` temporarily exits the input mode, and displays a list of all the available command names. You can then point to and browse one of the names to complete execution of the command.

You can execute **C-G** to abort execution of the command at any point during typing of the string.

String Completion When Executing M-X Commands
When executing M-X commands, the system can assist you in typing the command name. For example, executing **M-X**,

```
    nm [Return]
```

is enough to complete execution of the command that lists the facilities at the top level of NMODE (**M-X nmode root**). When [Return] is pressed, the message area completes the command name, `Nmode Root`. You can disregard case in typing the command name. This feature is called **completion**. You can press [Return], press the space bar or [ESC], or type [?] at any point in typing the string, depending on the kind of completion you want to do.

If you press [Return] at any point in typing the string **and** the partial string is unique to a particular available command, input terminates and the command is executed. Otherwise, pressing [Return] will only complete typing the command name as far as it can; the bell will beep to indicate an ambiguity. For example, if you execute **M-X**, type `li`, and press [Return], an "s" will be added to the string, but nothing else happens because the system cannot tell if you want a command that starts with "Lisp" or "List".

On pressing the space bar, the M-X command attempts to complete the string you are typing. If the string is several words such as `auto fill mode` and you type something like au before you press the Spacebar, then repeated pressing of the Spacebar can supply the remaining words: `Auto`, `Fill`, and `Mode`.

You can press the space bar at any time during the entry of an M-X command name. For example, after executing **M-X**, you might type:

```
    help
```

and press the space bar. NMODE will capitalize the word, but it cannot complete the string because there are several acceptable possibilities. If you continue by typing:

```
help dis
```

and press the space bar, the system completes the string and displays:

```
Help Dispatch
```

You can now press Return to execute the command if you really want to execute the indicated command. Otherwise, you can backspace and enter a different string.

What pressing the space bar does is let you see the rest of the string you are typing provided the string has some validity. If you type something like **glutenburger**, a string that has no validity in the environment, pressing the space bar takes no apparent action.

To get specific help, you can type ? at any time while entering the command. If you type something like **augo?**, a M-X command does not accept the question mark because the partial string has no validity within the system. You would need to backspace and type another string or abort the command with **C-G**. On the other hand, if you type something like **tran?**. NMODE temporarily suspends execution of the M-X command name and invokes a Prompt Browser major mode and shows only the command names related to the partial string. In this case, the display looks like this:

```
M-X command Name:
   (Browse item to complete command.  Quit to return to command.)

   Transpose Characters
   Transpose Forms
   Transpose Lines
   Transpose Panes
   Transpose Paragraphs
   Transpose Regions
   Transpose Sentences
   Transpose Words



I/O @ Browser (Input Prompt) Prompt

Help  Browse  Group  Filter  Quit
```

You can point to a command name and execute `Browse` to complete execution of the command, or execute `Quit` to get back to the extended command to continue typing the string. Note that softkeys, M-X commands, and most system commands are not available while in the Prompt Browser mode; NMODE forces you to complete or abort the command that is pending.

Aborting a M-X Command

Executing the Nmode Abort command, **C-G**, after you execute **M-X** and before you press Return during entry of a command name cancels the command. By the way, executing **C-G** also cancels other commands that require user interaction.

Special Commands in Menus

When you are in a browser mode, typed characters are not inserted because the mode write-protects the browser's items. This means that, in addition to executing applicable NMODE commands, you can execute special commands that do not require a prefix. These commands appear in a command line, which appears in the message area.

Press the key (shifted or unshifted) corresponding to the capitalized letter in the command's name to execute a command from the command line. Appendix B contains an alphabetical list of browser commands and describes their actions. If you see a command in a menu and do not know what it does, you can use the alphabetical listing in Appendix B to find the command and determine how it works. You can also get information by requesting Help and executing `Where-am-I` (**C-? W**).

Softkey Commands

The HP 46020A keyboard has two groups of softkeys along the top row: f1 through f4 and f5 through f8. NMODE uses these softkeys. In addition, the HP 46020A keyboard has an unlabeled set of keys in the upper-right corner of the top row, which NMODE does not use. The conventional softkeys look like this:

f1 f2 f3 f4   f5 f6 f7 f8

and a corresponding set of softkey labels might look like this:

```
Help    HP-UX    Nmode    Nmode     Buffers   Direc-   Find    Find
       Command  Command   Root                tories   File   Direct.
```

Note in the illustrations that `Nmode Root` is the softkey label for the f4 softkey, which executes the Browser Browser command. By pressing the softkey, you do not need to type the **M-X** prefix and then type the string, `nmode root`.

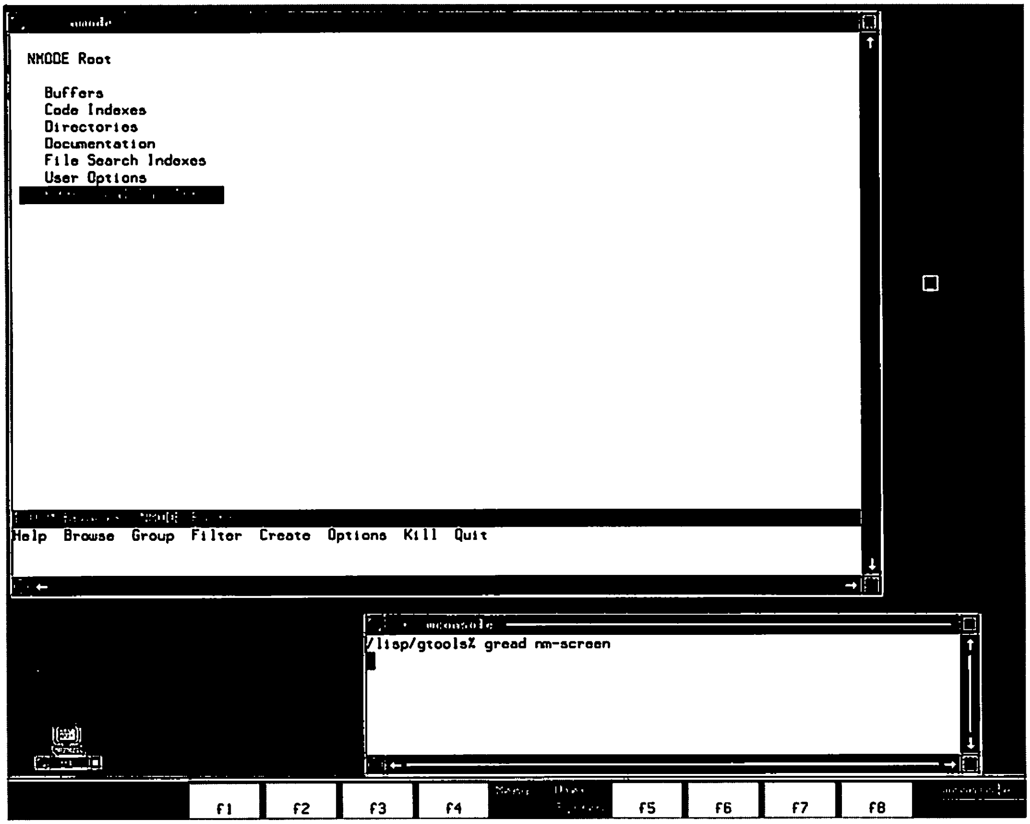
Appendix C, "NMODE Softkey Labels", contains an alphabetical list of all default softkey bindings for the NMODE user environment.

You can change the softkey bindings to suit your needs. See the "Customization" chapter for details about altering softkey bindings.

Locator Device Commands

A pointing device (locator device) is not required to effectively use NMODE. Any locator device supported by Windows/9000 can be used with NMODE, although the most useful devices are x-y locators with at least two buttons (mouse, or tablet with a puck). If your locator has only one button (a tablet with a stylus for instance), it will be used by NMODE as the left button, and you will not have access to the functionality of the right button. If your locator has more than two buttons, the extra buttons will be ignored by NMODE. If you do not have a locator, you may invoke left button commands with Select, and move the pointer by holding down CTRL while pressing the appropriate arrow key.

Be aware that all discussion relates to positioning the pointer on the screen item or inside the area being discussed and pressing the appropriate button on the device. Use the following screen display to help you determine what happens in the sections that explain how to use the left and right buttons.


NMODE Root

Buffers
Code Indexes
Directories
Documentation
File Search Indexes
User Options

```
/lisp/gtools% gread nm-screen
```

A Typical Screen With window-management Software

The Left Button

To use the left button, position the pointer "on" or "in" an item and press (click) the button to take the indicated action. Refer back to the typical screen as necessary.

- **Outside any window:** Invokes a window-management system-wide menu. This area is sometimes called the **desktop** or background.
- **On corner icons of a window:** Provides these options:
  - `Move` moves a window (the square with the diagonal arrow).
  - `Go Iconic` causes the current window to become an icon and makes a different window current (the empty square).
  - `Re-size` causes a window to change size. NMODE automatically adjusts itself. Windows cannot be made larger than their original, created size (the square with the enclosed, dotted square).
  - `Pause` pauses output to a Term0 HP-UX window. A second "click" resumes output (the octagon). Clicking on the pause icon has no effect on windows created by NMODE.
- **Inside the top, right, and bottom border areas of a window:** Invokes a menu for the window-management software. Only the currently available commands are highlighted and selectable.
- **Inside a non-active window:**Becomes the active window for both NMODE and the window-management software.
- **Inside a non-active pane in the active window:** Becomes the active NMODE pane.
- **On a softkey label:** Executes the corresponding softkey command.
- **In an NMODE item for browsing in the active window and pane:** Provides these options:
  - Moves the highlight to the pointed-at item. "Clicking" the current (pointed to) item browses the item.
  - On a browser command, executes the command as if you had pressed the capital letter in the command's name. This works for first and second level menus of browser commands.
- **In an editing buffer in Emacs mode in the active window and pane:** Provides these options:
  - Within any text, "clicking" either positions the cursor "under" the arrow or sets a mark if the cursor is already "under" the arrow.

- In the message area, "clicking" executes command under the pointer.

- **In input mode (during prompt for file name, extended command name, etc.):**
  - Clicking in message area generates a $\boxed{\text{Return}}$ to terminate input.
  - Clicking anywhere in the active window will abort the command (like C-G).

Right Button
Position the arrow in or on the "item" and press the right button to take the indicated action. Refer back to the typical screen as necessary.

- **Outside any window:** Provides a popup menu called `NMODE GENERAL`. See the later section called "Popup Menus" for details.
- **Inside the current NMODE window:** In a buffer, provides a mode related popup menu; in a browser, provides a popup menu that duplicates the menu of browser commands. See the later section called "Popup Menus" to get some details. Note that popping up a menu in a non-cative pane will not select that pane. This allows you to use the popup menu without obscuring the active pane.
- **Inside the right border of the active window:** Provides these options:
  - `Up-arrow` scrolls to the previous page.
  - `Down-arrow` scrolls to the next page.
- **Inside the bottom border of the active window:** Provides these options:
  - `Left-arrow` scrolls the window to the left (moves the cursor left).
  - `Right-arrow` scrolls the window to the right (moves the cursor right). When a horizontal scroll to the right occurs, an `>XX` indicator appears in the mode line, where `XX` is the column position of the leftmost displayed character.
- **Inside a non-active window:** Becomes both the NMODE window and the window-management software window.
- **In input mode (during prompt for file name, extended command name, etc.):**
  - Clicking in message area generates a $\boxed{\text{Return}}$ to terminate input.
  - Clicking anywhere in the active window will abort the command (like C-G).

Popup Menus

When you are in a pane, clicking the right button provides a popup menu of commands that relates to the current mode. In a buffer, the commands generally relate to the current minor mode within Emacs mode. In a browser, the commands generally duplicate the menu of special browser commands. The exceptions are that commands called `Help` or `Quit` might be available and there is a special command called `Nmode General` (more on this shortly).

When you are in the gray (desktop) area, clicking the right button provides a popup menu called `NMODE GENERAL`, which contains several commands. When you click on `Nmode General`, or click the right button in a gray area, you get the following popup menu:

```
NMODE GENERAL

Window Cmds >>
Places >>
Utilities >>
```

`NMODE GENERAL` is the name of the menu, and you can point to and click any of the commands: `Window Cmds.`, `Places`, and `Utilities`.

The string >> appended to a command in a popup menu indicates that clicking on the command with either button brings up another popup menu instead of taking a direct action.

In general, the commands provide a selected set of commands for manipulating windows, going to "places", or using certain utilities, respectively. The commands in the second level menus are straightforward. For example, the second level menu for places looks like this:

```
PLACES

Next Window
Next Pane
Nmode Root
Buffers
Directories
Documentation
User Options
Add. Facilities
Quit
```

The description of popup windows is deliberately brief because they provide a convenient but not essential way to execute commands. In addition, most commands provided in popup menus are straightforward. See the documentation for the window management software to get information not specifically related to NMODE.

NMODE's Items and Modes

This section discusses items and modes fundamental to NMODE. Because modes are important, the discussion has some deliberate redundancy so you will see the overall information in several contexts.

It was stated earlier that NMODE is a moded user environment. Modes control:

- The meaning of what you see on the screen.
- Which operations can be done.
- The use of things like the cursor and point, universal arguments, and procedures for aborts and exits.

The next subsection should clarify what a mode is, and the following subsections should provide additional information about how to use them.

The Items and Modes

A mode is a data structure that controls the behavior of NMODE. Each particular facility of NMODE, such as editing buffers, facility browsers, has an associated mode that defines the capabilities of that facility.

When you use a certain facility or item, NMODE knows the structure, state, and mode that the facility or item should have. For example, NMODE "knows" that the top level of NMODE (NMODE root) has a browsing mode in which you see a list of facilities that you can point to, browse, and otherwise manipulate. NMODE also knows that the facilities in the list are write-protected and that NMODE is not in a character-insert state. In this state, NMODE knows that it provides a menu of special browser commands.

On the other hand, when you enter an editing buffer, NMODE "knows" that a a list of items and a menu of browser commands are not presented. Instead, NMODE knows it should provide an editing buffer in which you can write or edit text, perhaps the text contained in an associated file.

Whether you invoke a browser for a facility or item, or you invoke an editing buffer, NMODE "knows" the associated mode that the browser or buffer should have. The associated mode determines the set of bindings between the keys on the keyboard and a set of NMODE functions that execute commands. The set of bindings changes automatically when you change modes. Check the **mode line** to find out what mode or combination of modes is current. The set of bindings between keys and functions is called a **dispatch table**. Later, you will learn commands that let you examine the bindings in a dispatch table.

The current mode affects the screen display in the following ways:

- When the current facility or item has a browser mode, you see a list; you can execute both NMODE and menu commands; and you cannot insert characters directly into the items in the list.
- When the current item (typically a buffer) has an editing mode, you do not see a list; there are no command line commands; and you can insert characters directly into the item.

While the above information is generally correct, in the most fundamental context, a **mode** is a **binding** of keyboard keys to NMODE functions that implement commands. The binding is not limited to keyboard keys, and can include devices such as a mouse. You can execute the commands according to the alternatives provided by the current mode. Of the available major modes, the most common "appear" in the mode line as `Browser` or `Emacs`. Any associated minor modes appear inside parentheses just to the right of the major mode; for example, you might see:

```
Browser (Buffers)
```

which indicates you are in Browser mode with buffers minor mode, and the corresponding items in the list are buffers (workspaces); or you might see:

```
Emacs (Text Fill)
```

which indicates you are in Emacs mode with text and auto fill minor modes, and the items are the characters in the buffer that you can edit.

NMODE can have one major mode and several associated minor modes at a time.

You will have a better grasp of how NMODE works if, when viewing the mode line, you associate the mode for `Browser` with manipulating lists of items and you associate the mode for `Emacs` with writing and editing text. To restate this in a different way, keep the following information in mind as you learn more about using the NMODE environment.

- You are always using some type of item in some type of mode; for example, an active directory in a browser mode or a buffer in an editing mode. You execute commands to manipulate these items, and NMODE has about 300 commands for this purpose.

The mode changes automatically as you access a different type of item. This is a nearly transparent action. Just remember that NMODE provides the functionalities related to the current item. When you work with files, NMODE provides the mode for working with files by providing a certain set of commands. If you change items, and work with buffers, NMODE provides another mode that is suited to working with buffers by providing a different or altered set of commands.

The Cursor and the Point

In an editing mode, the current pane has a **visible cursor** and the related buffer has an **invisible point** that are closely related. In general, you pay close attention to the cursor during text editing; the system deals with the point (more on this shortly). The cursor is also important during user interaction in which you type entries related to prompts in the message area. In using a browser, you typically focus on a current item and might not even notice that there is a cursor, which can be moved along the highlighted line.

The **cursor**:

1. Shows on the screen the location for character entry in a buffer relative to the point when you edit text.
2. Sits somewhere in the highlighted region that points to the current facility or item in a browser's list. It usually sits at the left end, but can be moved.
3. Shows the location for interactive character entry related to a prompt or recursive editing related to a search. In these cases, the cursor appears in the message area.

In a buffer, an item called the **point** keeps track of the location for entry of characters according to line number and column. The point is not visible. The point lies immediately to the left of the cursor, but after the preceding character.

Because you can see the cursor, this manual usually refers to the cursor. The point is mentioned when it is necessary to be more explicit, especially in the chapters called "Working With Text" and "Working With Lisp Code". Think of the point as an "un-lighted" cursor.

Universal Arguments

You will often want to repeat a command a specified number of times or otherwise alter the functionality of a command. This is done by executing the Universal Argument command, **C-U <integer>**, which gives a command a **prefixed argument**. This means that you provide the argument **before** you execute a command and without typing any spaces in the total key sequence. For example, **C-U 23 C-F** means that you type or hold the following keys without typing any spaces.

CTRL-U 2 3 CTRL-F

Many NMODE commands accept a prefixed argument supplied by the Universal Argument command. The prefixed argument is an integer. In some cases, you can execute **C-U** and press Return without entering an argument. The Universal Argument command does **not** have an M-X command alternative; that is, you cannot execute **M-X universal argument**.

A universal argument can have the following general effects, depending on the command and situation:

- **Iteration**: In an iterative situation, a prefixed argument, **n**, executes a command **n** times. For example, **C-U 20 C-F** moves the cursor forward 20 spaces.
- **Reversed direction**: Most commands that have direction, such as the Move Forward Character command, reverse direction when given a negative universal argument. Negative arguments are typed with **C-U - <integer>**.
- **Altered Functionality**: For some commands, having an argument is important regardless of its value. Such commands only "care" about whether there is an argument, the presence of which alters the functionality of the command. For example, in a buffer in Text minor mode, the **M-Q**, Fill Paragraph command, fills text in a paragraph. With an argument, **C-U 23 M-Q** or just **C-U M-Q**, the command fills **and** justifies the text.

Numeric arguments can be positive, zero, or negative within the range accepted by a command. In most cases, a negative integer reverses the action of a positive integer. The **-1** or **0** integers can be special cases. Such special cases are mentioned in the *Comments:* entry in the *NMODE Command Reference* manual.

A sequence of **C-<digit>** arguments works the same way as an equivalent **C-U <integer>** argument. For example, **C-1 C-3 C-2** has the same effect as **C-U 132**. Besides **C-<digit>**, you can use **M-<digit>** and **C-M-<digit>**. C– (Control-minus) works the same as **C-U**

There are some additional special situations. For example, a **C-U** followed by nothing or something other than a minus sign or digit means "multiply by four". The following examples illustrate the idea.

- **C-U** d displays four ds.
- **C-U C-U C-F** moves forward 16 items; for example, 16 characters in a buffer or 16 buffers in the Buffer-Browser.
- **C-U 10 C-U C-F** moves forward 40 items.
- **C-U C-U C-U M-D** deletes 64 words.

Cancels, Aborts, and Exits

However you use the NMODE environment, you will encounter times and situations in which you want to abort or exit some state or item and go to another state or item.

The following commands let you abort, terminate, or exit commands, processes, states, or items.

C-G	The NMODE Abort command aborts execution of a command when you use an M-X command. The command works while you attempt to enter a command name, after you execute **M-X**, and before you press $\boxed{\text{Return}}$. By the way, **C-G** also "aborts" a key sequence in which you have a distinct prefix and a final character (e.g. **C-X Z**) if you execute **C-G** before you type the final character.
Quit	This special command in every browser's menu exits the current item and returns to the previous item. If you often moved from one browser to another during a current session, repeatedly executing `Quit` moves back through the browsers in an exact reverse-sequence because NMODE tracks your use of items throughout one session.
Lisp q	When you are in Lisp minor mode in Emacs mode and get into break loops (enter the Debugger), the Lisp Quit command exits one break loop and enters the next higher level, which might still be a break loop. The **Lisp-** prefix is **C-]**, so the key sequence is $\boxed{\text{CTRL}}$-$\boxed{]}$ $\boxed{\text{Q}}$.

Lisp a	When you are in Lisp mode in Emacs mode and get into break loops, the Lisp Abort command exits any number of break loops and returns to the top level. The **Lisp-** prefix is **C-]**, so the key sequence is CTRL-] A.
Lisp l	Exits the NMODE user environment and returns you to the underlying Common Lisp system in the HP-UX window that spawned NMODE, but not back to the HP-UX system. The **Lisp-** prefix is **C-]**, so the key sequence is CTRL-] L. You can type `(ed)` and press Return to get back to the NMODE user environment.
C-X Z	Exits the NMODE user environment and returns you to the HP-UX operating system.

Assorted Topics

As you work with the NMODE environment, you will probably encounter situations where you want to get back to some known state, get help, or use HP-UX. You will also need to know some methods for using the environment. The next few sections contain some basic information that is used in later chapters.

Useful Routines

This section describes three very useful methods (procedures) that you can continually employ as you read subsequent chapters and learn how to use NMODE. The routines are definitely part of the big picture for using the NMODE environment.

The Point and Browse Routine

This subsection describes the "point and browse" routine that was mentioned in some earlier sections. This is probably the best routine for inexperienced users. Therefore, it is fully described in a setting that you can hypothetically follow and subsequently apply to your situation.

Suppose you are at the top level of NMODE, you want to edit a file, and you see the following facilities:

```
NMODE Root

Buffers
Directories
Documentation
User Options
Additional Facilities
```

Suppose also that your file is in `Directories`. With this situation in mind, the following steps illustrate the **point and browse routine.**

Step 1

Press $\boxed{\blacktriangle}$ or $\boxed{\blacktriangledown}$ as required to move the highlighted region onto `Directories`. There are other ways to move the highlight that you will learn later. This step is called "pointing to" a facility or an item.

Step 2

Execute the browser command named Browse, which appears in the message area near the bottom of the screen. You execute any browser command by pressing the key (shifted or unshifted) corresponding to the capitalized letter in the command's name. This is the "browse" part of the routine. After executing Browse, you could see a list of active directories something like this:

```
Active Directories

$HOME/
/users/joe/l-code/
/users/joe/scripts/
```

At this point, be aware that the display could be blank, except for the title, Active Directories. The reason is that you need to create active directories under the facility called Directories. You probably did this in your NMODE initialization file. In any case, assume for now that the directories are available. A later chapter called "The Browser for Directories" explains how to create active directories.

Step 3

Press $\boxed{\blacktriangle}$ or $\boxed{\blacktriangledown}$ as required to move the highlighted region onto the active directory named /users/joe/l-code/. Again, this is the procedure that lets you point to an item, a directory in this case. Pointing to any available item makes that item the **current item.**

Step 4

Execute the Browse command to invoke the browser for the current (active) directory. You would see a list of files something like this:

```
Directory: /users/joe/l-code/
Last Read:  9-Oct-85 09:23:41     (Multiple-selection is ON)

             Directory    ..dir...   1-Oct-85 07:40:18 drwxr-xr-x
..           Directory    ..dir...   1-Oct-85 09:21:00 drwxr-xr-x
some-demo.l  Lisp Source       356  12-Oct-85 12:32:56 -rw-r--r--
some-demo.b  Lisp Code         746  13-Oct-85 14:12:31 -rw-r--r--
```

Ignore the information at the top of the display for now. It is explained later.

Step 5

Press $\boxed{\blacktriangle}$ or $\boxed{\blacktriangledown}$ enough times to move the highlighted region onto the file named some-demo.l and execute the Browse command. Now that you are down to a specific item, the command works differently. It does not just invoke a browser that produces another list of items. In this hypothetical case, executing Browse:

- Exits the active directory browser;
- Invokes an editing mode;
- Creates an editing buffer to hold the current file; and
- "Visits the file" in the created buffer.

You could then edit the file. The peculiar phrase, **visits the file**, means that the file is associated with a buffer and made available for editing.

While the exercise was hypothetical, this is the point and browse routine. It is a fundamental way to get around in the NMODE user environment that is friendly and easy to use.

You would probably want to save the edited file. The chapter called "Working With Text" explains the commands and procedures to do this. You would also want to get back to a part of the environment from which you could do other work. The section "Getting to a Known State" later in this chapter explains some fundamental alternatives. Several later chapters mention ways to move around in the NMODE environment.

To use this routine effectively, you need to have some notion about where a file is located, and it is helpful to know the name of the file you want to edit. If you have no notion about the location or name, you might need to systematically browse the contents of several directories or get to a shell and use the HP-UX *find(1)* or *grep(1)* commands to locate your file.

The Menu Driven Routine

Anytime you are in a browser, the message area provides a menu of commands for that browser. You can execute these commands in a **menu driven** routine according to how you want to manipulate the items in the browser's list. In many cases, a particular browser command provides a lower level menu of commands. Although the menus are not visibly apparent from the top level, this nesting of menus can reach down three or four levels. Appendix B, "Special Browser Commands", contains a chart that shows this.

The Direct Access Routine

While the point and browse routine is easy to use, it often lacks efficiency. The menu driven routine primarily lets you manipulate items in a browser's list; for example grouping and filtering items are common manipulations. This sections describes a **direct access routine** that lets you "branch" directly to an item.

Again, suppose you are at the top level of NMODE and you want to edit the file named `some-demo.l`.

In the direct access routine, you execute an NMODE command that takes you directly to a specified item. You need to know: the name of the command; the method of executing the command; and the specific name of the item you are accessing.

Continuing the hypothetical example, to directly access the file named `some-demo.l`, which is in the directory named `/users/joe/l-code/`, you could execute the Find File command via a "key sequence", which appears as **C-X C-F** in this manual. In the case at hand, you would use the following procedure:

Step 1

Execute the Find File command via the key sequence. The **C-X C-F** key sequence means: hold down [CTRL] and press [X], release both keys, hold down [CTRL] again and press [F], and release both keys. This initiates execution of the command that will find the file.

Step 2

You will be prompted to enter a file name relative to the current directory; for example, the prompt in the echo area might look like this:

```
Find file: (relative to '/users/joe/')
```

Since the file you want is in the directory named `l-code` under the relative path in the prompt, you would type:

```
l-code/some-demo.l
```

and press [Return]. The Find File command would then exit the current mode, find the file, create a buffer, and visit the file in the created buffer. You could then edit the file.

Notice that the direct access routine is more efficient, but you must know specific types of information. It may take some time to memorize the commands and procedures for using it. Once you learn to use NMODE's commands, the direct access routine provides a powerful and efficient way to use the total environment.

Getting to a Known State

When you get into some unknown state or location and have some doubts about how to recover, you can always execute one of the following commands to get back to a known location and state.

C-X R	The Nmode Root command, also available as **M-X nmode root**, takes you to the top level on NMODE from which you can then access any facility in the total environment.
C-X C-B	The Buffer Browser command, also available as **M-X list buffers**, takes you to the browser for `Buffers`, which list buffers for writing and editing text and provides a menu of commands.
C-X D	The Browse Directory command, also available as **M-X find directory**, takes you to any specified active directory.

More is said about these commands later. They are mentioned here because they are commands that people use to get to familiar departure points.

Getting Help

NMODE provides various commands and online manuals that provide online help. You can directly point to and browse information items or execute commands that provide information. The procedures, and commands for getting help are explained later in a chapter called "Getting Online Help". In addition, Appendix A has a list of commands for getting help.

The HP-UX System Access Facility

The optionally loadable HP-UX System Access facility lets you interact with the underlying HP-UX operating system without leaving the NMODE environment.

When you are in the NMODE user environment and wish to execute HP-UX commands, the HP-UX System Access facility provides two interfaces and a special command.

- An HP-UX shell-buffer interface provides a default shell-buffer whose name is HP-UX.SHELL. A shell-buffer lets you directly execute many HP-UX commands and most NMODE commands.
- A general HP-UX system command interface lets you execute commands through NMODE's system shell from any buffer in Emacs mode. This interface also provides the HP-UX Execute By Prompt command, **C-X H**, which works at any time in any mode (browser or buffer), and can also be invoked by the "HP-UX Command" softkey when the system shell is available.

The HP-UX access facility provides direct communication with invocations of HP-UX shells. The details about using the HP-UX Facility are provided in the later chapter called "The HP-UX Facility".

Conclusion

This chapter contained fundamental information about such things as the screen display, browsers, buffers, modes, facilities, routines, and so on. This information is sufficient for you to read any additional chapters and learn how to use some part of the NMODE user environment.

You need not read the chapters sequentially. If you want to know assorted ways to get help, read the chapter about getting online help. If you want to use the HP-UX System Access Facility, read that chapter. The only recommended sequences are that you read the introduction to browsers before you try to work with a specific browser, and read the chapter about working with text before you try to write source code in Lisp, C, Pascal, or Fortran.

Chapter 3
NMODE Windowing

Introduction

In addition to utilizing the Windows/9000 libraries, NMODE supplies windowing capabilities of its own. For instance, within a single Windows/9000 window, NMODE can divide the screen into multiple areas (**panes**), each one of which may display a different part of the NMODE environment. This chapter defines NMODE windowing concepts and describes how to access these windowing capabilities.

Concepts

The two most important terms for discussing NMODE windowing are **window** and **pane**. Remember that the definitions given here apply to NMODE; they are not necessarily consistent with the way these words are used in other manuals.

Windows

An NMODE window is a virtual display device that is used for output from NMODE. There are two basic types of NMODE windows:

1. Ones that are Windows/9000 graphics windows. These are used when NMODE is invoked (without the `-t` option) while the Windows/9000 window manager is running. The number of these windows available at one time is limited only by Lisp heap size and Windows/9000 resources. We will refer to this type of NMODE window as a **first-class** window.
2. Ones that are terminals, term0 windows, or external displays. If you invoke NMODE with the `-t` option or without the window manager running, you will be using this type of NMODE window. A 98700 used as an auxiliary display device for NMODE is also this kind of window. These are **second-class** NMODE windows.

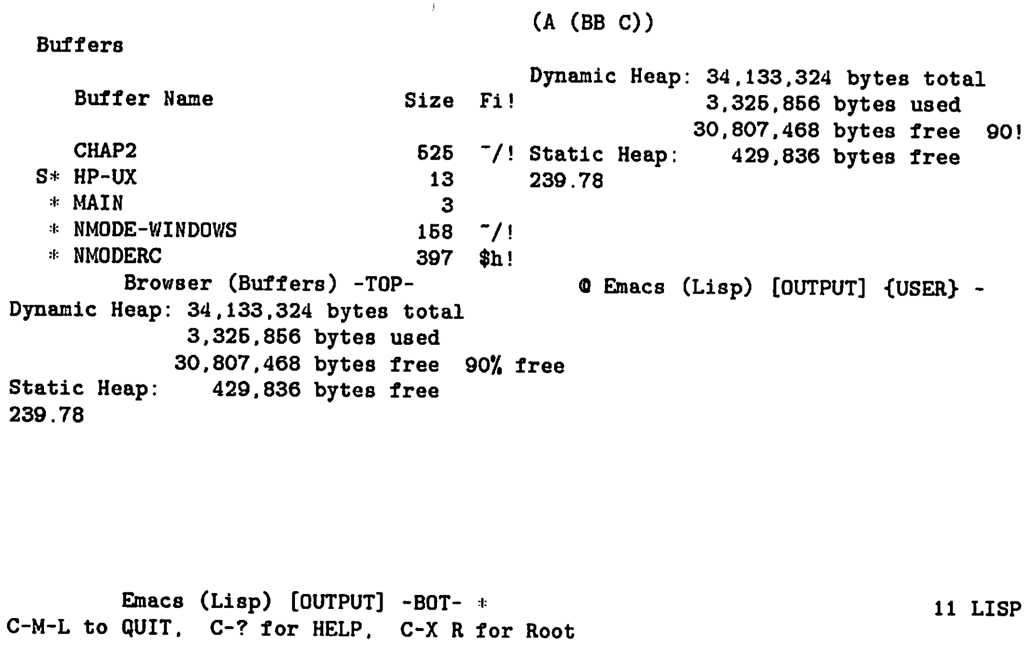
At a particular time, you may have several first-class windows, and one second-class window corresponding to a 98700 auxiliary display. If you invoke NMODE from a terminal device with the `-t` option or when the window manager is not running, you can have at most two NMODE windows: one corresponding to the terminal, and an optional one on a 98700 auxiliary display. Both of these are second-class windows.

In the latter case, when you have only second-class windows, there are fewer features available. You cannot use the mouse; left [Extend char] does **not** work as Meta; the numeric keypad keys are not distinguished from their regular counterparts. For more information about these restrictions, consult the *Installation and Overview* manual and Appendix D of this manual.

NMODE always has at least one window.

Panes

An NMODE window can be divided into one or more **panes**. Each pane has its own mode line information and can display a different part of the NMODE environment. Figure 3.1 shows an NMODE window with three panes. The upper- left pane is showing the Buffers Facility; both the upper-right pane and the lower pane are showing the OUTPUT buffer.



```
                                                  (A (BB C))
  Buffers
                                                  Dynamic Heap: 34,133,324 bytes total
     Buffer Name                 Size  Fi!                       3,325,856 bytes used
                                                                30,807,468 bytes free  90!
     CHAP2                        525  ~/! Static Heap:    429,836 bytes free
  S* HP-UX                         13      239.78
   * MAIN                           3
   * NMODE-WINDOWS                158  ~/!
   * NMODERC                      397  $h!
         Browser (Buffers) -TOP-              @ Emacs (Lisp) [OUTPUT] {USER} -
Dynamic Heap: 34,133,324 bytes total
               3,325,856 bytes used
              30,807,468 bytes free  90% free
Static Heap:    429,836 bytes free
239.78




        Emacs (Lisp) [OUTPUT] -BOT- *                                   11 LISP
C-M-L to QUIT.  C-? for HELP.  C-X R for Root
```

Figure 3.1: An NMODE Window

- A single existing pane can be split into two panes with either a horizontal or vertical border.
- At a particular point in time, there is one and only one **selected pane**. This is the pane that is receiving NMODE's input from the keyboard and responding to the commands you type. The selected pane is indicated in its mode line with the string contained in the special variable `nmode::nmode-selected-window-symbol`, which defaults to " @ ". In Figure 3.1, the upper-right pane is the selected pane.

Commands and Techniques

This section describes ways to create and manipulate NMODE windows and panes. Most of the windows commands described apply only to first-class windows. In the *NMODE Command Reference*, the commands discussed here appear in the "Windows and Panes" section.

Window Commands

Creating

If you already have at least one first-class window, you can create additional ones while NMODE is running. The default characteristics (size, location, font, etc.) of the window you create are controlled by NMODE Window Creation Options in the User Options Facility. There are two ways to create a window with the default characteristics:

- Execute **M-X create default window**.
- Browse the `Create window` field of the Window Creation Options browser.

There are two ways to create a first-class NMODE window with characteristics different from those specified in Window Creation Options:

- Execute **M-X create window**. You will be prompted to enter each of the window's characteristics one at a time. For any particular characteristic, you can accept the default by just typing Return.
- Evaluate a call to the function `nmode:create-terminal` with the first argument being `:window`. Such a call will look like:

```
(nmode:create-terminal :window
                     :name string
                            :fonts string-or-list-of-strings
                            :x-origin number-of-pixels
                     :y-origin number-of-pixels
                            :width number-of-characters
                            :height number-of-characters)
```

  Fonts is a string or a list of strings for the desired font file(s) relative to `/usr/lib/raster`. If you omit one of the keyword parameters, the default value for that characteristic will be used.

Selecting

The window containing the selected pane is the selected window. To select a pane in another window you must first select the window containing that pane. You can think of all the available NMODE windows (first and second-class) as being in a circular ordered list. To select the next window in the list, execute the Select Next Window command, **C-X N**. To select the previous window in the list, execute the Select Previous Window command, **C-X P**.

You can select an arbitrary NMODE window with the Select Window command, **C-X W**. You will be prompted for the name of the window to be selected.

If your locator is enabled, you can select a first-class window by positioning the pointer over the window you wish to select and pressing either locator button.

Killing

First-class NMODE windows can be killed in a number of ways. If you try to kill your last first-class window you will be prompted for confirmation that you really want to exit NMODE.

There are several ways to kill a first-class window:

- Use a Windows/9000 popup menu to destroy the graphics window corresponding to the NMODE window you wish to delete. If you use this method on your last window, you will **not** be prompted for exit confirmation.
- Execute **M-X kill window**. You will be prompted for the name of the window to be killed.
- Select the `Kill` item from the NMODE window commands popup menu.

Changing Size

The `Size` item of a Windows/9000 popup menu or the lower-right border icon can be used to change the size of a first-class NMODE window. The mode line will be repositioned and resized to fit the new size of the window. You cannot make an NMODE window larger than its original size.

Moving

The `Move` item of a Windows/9000 popup menu or the upper-left border icon can be used to move a first-class NMODE window.

Hiding (Changing to an Icon)

The Icon item of a Windows/9000 popup menu or the upper-right border icon can be used to change the size of a first-class NMODE window. There is a special "HP-AI" icon for first-class NMODE windows.

Writing a Window's Contents

The contents of the selected window can be written to a file by executing **M-X write window image**. You will be prompted for the name of the file to which the window's contents will be written.

Pane Commands

This section describes commands for manipulating panes.

Creating

Initially, a window displays one pane. This (or any other) pane can be split either horizontally or vertically to create a new pane.

To split a pane into two panes with the same width but half the height as the original pane, execute the Create Lower Pane command, **C-X 2**. The selected pane is split and the newly created pane becomes the selected pane.

To split a pane into two side-by-side panes half the width of the original pane, execute the Create Side Pane command, **C-M-2**. The selected pane is split and the newly created pane becomes the selected pane.

Selecting

Each NMODE window has an ordered series of panes. To select the next pane in the list, execute the Select Next Pane command, **C-X O**. If the current window only contains one pane, executing this command replaces the selected pane with an alternate one.

You can use the locator to select a specific pane of a first-class window. The window must first be selected. Then position the pointer over the pane you wish to select and press the left locator button.

Killing

To kill the current pane, execute the Kill Pane command, **C-X C-K**. The pane disappears and the next pane is selected.

To kill all a window's panes except for the selected one, execute the View One Pane command, **C-X 1**. All but the selected pane disappear, and the selected pane occupies the entire window.

Changing Size

There are four commands that change the size of a pane. Which one to use (and its effect) depends on the pane you wish to alter. The commands are:

- Grow Pane Down command, **M-X grow pane down.**
- Grow Pane Up command, **C-X ^**.
- Grow Pane Left command, **C-M-<**.
- Grow Pane Right command, **C-M->**.

All of these commands works as follows: If the pane is bordered by another pane in the direction it is to grow, the pane is grown one line or column larger in that direction (the bordering pane is shrunk by a line or column). Otherwise, if the pane is bordered in the opposite direction by another pane, it is grown in that direction. Otherwise, nothing happens.

You can use the locator to change the size of a first-class window's panes. Position the pointer over the pane border (mode line or vertical border) you wish to move and press the right locator button. Position the pointer at the new location for the border and press either button. The border will be moved, effectively resizing one or more panes.

Transposing

You can switch the positions of any two panes in a window. First mark one of the two panes by selecting it and then executing the Mark Pane for Transpose command, **M-X mark pane for transpose**. Then select the other pane that is to be transposed and execute the Transpose Panes command, **C-X E**. Each pane that is switched assumes the position and size of the pane it was switched with.

Window Enhancements

You can change the display characteristics of a particular NMODE window to suit your personal preferences. The only NMODE windows that cannot be modified in this fashion are ones associated with a terminal device.

Any other NMODE window is divided into six areas: background, text, browser line, vertical border, mode line, and lisp listener prompt. Each of these areas except background can have a unique font (all must be the same size), color, and video enhancement. The background area can have only a color enhancement.

Fonts

When an NMODE window is created, it is given a list of available fonts (all the same size). By convention, windows have a normal, bold, and italic font. You can control which of these fonts is used to write characters to a given area of the window. Be aware that the only font size that supports all three styles is 8x16. To see what styles are available for a particular size, check the possible `Window font` values in `NMODE Window Creation Options`.

Colors

If you have a color display, you can change the color used for each window area. Your display driver determines the number of colors that are available, but NMODE can use only the first sixteen on the display's color map.

Character Enhancements

The characters displayed in each area can have a unique enhancement. The available enhancements are underlining and inverse-video.

Setting up the Enhancements

You can set the enhancements for an NMODE window with a call to `nmode:set-screen-enhancement`.

`(nmode:set-screen-enhancement` *area window font color highlight*`)` *Function*

area	The area of the window for which you are specifying the enhancement. It can be `:background`, `:text` (the main area of the window), `:browser-line` (the line in a browser that indicates the current item), `:vertical-border` (the border between side-by-side panes), `:mode-line`, or `:lisp-listener-prompt` (the right-hand side of the OUTPUT buffer's mode line).
window	A string containing the name of the NMODE window for which you are setting the enhancements. If you're running NMODE without Windows/9000, the name of the window on the console display is `main-screen`. The name of the window on a 98700 used as an auxiliary display is `/dev/crt98700`.
font	One of `:normal`, `:bold`, or `:italics`. If the window does not have the desired font available, the normal font is used instead.
color	An integer index into the window's color map, or one of `:black` (0), `:white` (1), `:red` (2), `:yellow` (3), `:green` (4), `:cyan` (5), `:blue` (6), or `:magenta` (7).
highlight	The type of character enhancement to be made. Should be either `:underline`, `:inverse-video`, or `:normal-video`. An argument of `nil` is the same as `:normal-video`.

Example

Here's an excerpt from an initialization file (.nmoderc) showing how you might choose to set up your window enhancements.

```
(nmode::set-screen-enhancement :browser-line "nmode1" :bold :white :underline)
(nmode::set-screen-enhancement :text "nmode1" :normal :white nil)
(nmode::set-screen-enhancement :lisp-listener-prompt "nmode1" :bold :white nil)
(nmode::set-screen-enhancement :mode-line "nmode1" :bold :white :inverse-video)
```

Windows/9000 Interactions

Since NMODE uses Windows/9000, it is useful to know some of the specifics about the interactions between the two. This section lists some important information about NMODE windows and their relation to Windows/9000.

- When the window manager is running, and NMODE is not invoked with the -t option, NMODE windows are retained-raster graphics windows.
- NMODE only uses the first two buttons of locator devices. It will ignore any others.
- The effects of the WMIUICONFIG environment variable are restricted:
    - Locator buttons are presently "hard-wired" and cannot be configured.
    - NMODE windows ignore the "top on select" bit (0x100).
    - NMODE windows ignore the bit that disables the border scroll arrows. (0x10000).
    - NMODE windows ignore the bit that disables pop-up menus over the desk top (0x40000).
    - NMODE windows ignore the bit that turns off the beeper for aborted actions (0x4000000).
- WMBASEFONT and WMALTFONT are ignored.
- The window system environment variables for color (WMDESKFGCLR, WMDESKBGCLR, WMBDRFGCLR, and WMBDRBGCLR) are ignored.
- WMLOCSCALE is ignored.

Chapter 4
Getting On-line Help

Introduction

This chapter discusses how to get on-line information (help) within the NMODE environment. You can get help in these ways:

- If you just want to examine on-line documentation, browse into `Documentation` from `NMODE Root`. You can then choose the on-line documentation you want to browse.

- Anytime you need help during a working session, you can invoke a menu of commands that let you access the on-line Help facility. This form of help is available from any mode/location except input mode or prompt browser mode (which is an extension of input mode).
- At those times when you get into an input mode (e.g. in responding to a prompt for input), you can get help specifically related to the current situation.

Note that all routines for getting help use the documentation items in the Help Facility called `Documentation`.

Documentation Facility

Nmode Root contains the facility called `Documentation`. This facility contains all of NMODE's on-line documentation in browsable form. When you want to examine the on-line documentation, you can execute **M-X Nmode Root** from any place in the NMODE environment and then you can browse the Documentation Facility.

The Documentation facility contains items that you can browse to view on-line documentation. The default contents of this facility are listed below:

```
Documentation (a top level on-line information facility)
```

Environment Tutorial	(a "hands-on" tutorial of the environment)
NMODE Commands	(a list of commands and their description)
NMODE Glossary	(a list of terms and their description)
NMODE User's Guide	(the on-line version of the manual)
Lisp Reference	(a list of commands and their description)
Lisp Functional Guide	(the on-line version of the manual)
Lisp Programmer's Guide	(the on-line version of the manual)
Lisp Application Notes	(the on-line version of the manual)
Installation and Overview Guide	(the on-line version of the manual)

Documentation Facility Contents

There are three types of documentation provided: the tutorial, help items used by other parts of the help system (NMODE Commands, NMODE Glossary and Lisp Reference) and manuals (NMODE User's Guide, Lisp Programmer's Guide, Lisp Application Notes and Installation and Overview Guide).

The Documentation Facility does not contain the contents of all these manuals by default. If an item is loaded, an `L` is placed next to it in the list.

When you browse an item that is not loaded, NMODE prompts you with:

```
Documentation not yet loaded, do you wish it to be? (Default is: 'Yes')
```

Pressing [Return] causes the system to load the documentation item you want to browse.

The next three sections describe the three types of items contained in the Documentation Facility: Tutorials, Help Items and On-line Manuals.

The Tutorial

The Environment Tutorial provides interactive tutorials on many NMODE features. To use the tutorial, browse the Documentation browser from NMODE Root. Then browse the Environment Tutorial.

Unlike the manual type documentation, the tutorial is a program which uses softkeys and controls the display windows. When you browse the tutorial, two panes are shown side by side. In the right pane a list of tutorials available is shown. In the left pane are instructions for using the tutorials.

```
Environment Tutor

            How to Use NMODE Tutorials
            NMODE Command Interface
            Help
            NMODE Facilities
            Directory and File Access
            Emacs Editor
            Lisp Development
            Windows
            HP-UX Access
            Language Editing
            Code Indexing
            Error Indexing
            Search Indexing
            NMODE Customizations
```

The first time you use the tutorials, it would benefit you to browse the How to Use NMODE Tutorial first. This tutorial describes the softkeys and windowing system used in the tutorials.

You exit the tutorials by pressing the softkey **Leave Tutorial**.

NOTE

You have not "exited" the tutorial until you press **Leave Tutorial**. (Quit works in the tutorial browser also.) You cannot re-enter the tutorial once you are in it.

Help Items

You can browse the Help Items used by the help facility. These include the NMODE Commands, NMODE Glossary and Lisp Reference. The NMODE Glossary is used in the following discussion. You can browse the other items in a similar manner.

When you browse the glossary from the Documentation level, all the glossary items are listed. When you browse one of those items, a frame of information on the glossary name is displayed.

For example, browse the NMODE glossary. A list of glossary items appears on the display.

```
Documentation: Nmode Glossary

ABORT                                      glossary item
ACTIVATE DIRECTORY                         glossary item
ACTIVE/ACTIVATE DIRECTORY                  glossary item
BINDING                                    glossary item


  .
  .
  .
```

Now browse ABORT. The definition of abort is displayed. This is called a frame of information.

```
abort

Some commands perform a simple operation with no user interaction.  Other
commands provide for user interaction or performs a series of operations.
The user can abort these commands by executing a command that aborts an
interactive operation or gets the user out of a certain state in a series
of states.
```

To see another glossary item, return to the list by typing **Q**, then select and browse the desired item.

On-line Manuals

The on-line manuals are the text of the manuals placed in files that you can read. When you browse a manual, the table of contents is displayed. For example, browse the Lisp Programmer's Guide. The table of contents for this manual is displayed.

```
Chapter 1.  Introduction                                1:0
Chapter 2.  Concepts                                    2:0
Chapter 3.  Programming Tips                            3:0

    :
```

When first displayed, two levels of filtering are in effect. This reduces the number of headings shown. To see the sub headings and minor headings, execute Filter Undo. To return to only the chapter headings, execute Filter Re-do.

The number shown next to the heading indicates the file number and line within the file where the heading appears. When you browse a heading, you are placed in that file at that line number. The entire chapter is made available by this action so you may look at all the pages in the file. To go from one chapter to another, however, you must quit the text-browsing and return to the table of contents. There you can select another chapter to browse.

Help Via Menu Commands

This section discusses the menu driven routine that lets you use the Help facility to get detailed or focused help during a work session. If need a quick reference, "Appendix B: Browser Commands" has descriptions of the special Help facility commands.

Using the Help Command

All browsers have a command menu which includes a command named `Help`. Executing `Help` provides a menu of Help commands. Each command at this level provides a certain type of help. Some of the commands provide another level of commands. This nesting of commands does not exceed three levels to give you the help you want quickly. Think of getting help within the Help facility as temporarily suspending what you are doing and browsing down as necessary to get focused help.

If you are not in a browser mode and the command menu is not available, the NMODE Help Prefix, C-? (or C-/ so you don't have to use the Shift key), invokes the Help mode and provides the same menu of help commands.

Executing Help in Any Browser

All browsers contain a menu of commands and every top-level menu has a command named `Help`.

Executing `Help` invokes the On-line Help facility, which provides a menu of lower-level commands. These commands provide a menu driven routine for getting help. If you are not in a browser, executing **M-X Help** or C-? invokes the same help facility.

The following top level help menu is displayed.

```
Where-am-I Explain Key-bindings Documentation Tutor Help-help Quit-help
```

Each command works as follows:

`Where-am-I`	Displays a description of the current mode or application, and provides brief descriptions of available commands.

`Explain`	Prompts for entry of a string, the default is the item closest to the cursor before the command was given. The command then searches for information items related to the string and presents a browsable list of matching browsable items. For more information on the Explain command, refer to the NMODE Command Reference.
`Key-bindings`	Provides a menu of commands that are discussed later in the next section called *The Key-bindings Command.*
`Documentation`	Invokes the facility called `Documentation`, which provides access to the on-line manuals as well as other information.
`Tutor`	Invokes an on-line interactive tutorial.
`Help-help`	This second level of help provides a description of the top level help commands.
`Quit-help`	Exits the help facility and returns to your most recent location in NMODE.

The Key-bindings Command

This command provides the following menu of commands.

```
  Name  Description  All-commands  Help-help  Leave-help  Quit-key-help
```

Here are descriptions of each command.

- `Name`: The command prompts for entry of a key sequence or **M-X** command name, and then briefly displays the name of the bound function in the current mode. On an invalid entry, the command indicates that no binding exists.
- `Description`: The command prompts for entry of a key sequence or **M-X** command name and then displays related descriptive information in the current pane. Current alternate bindings to the function are also shown. On an invalid entry, the command indicates that no information is available.
- `All-commands`: This command outputs the dispatch table. The output shows the bindings between functions, key sequences, and **M-X** command names in the current mode. Here are the commands.

  - `Screen`: Displays a read-only list of bindings in the current pane. The menu of commands lets you get more information about an item, or quit the listing.
  - `Buffer`: Prompts for entry of a buffer name, and then appends the list of bindings for the current mode to the buffer, creating a buffer if necessary.
  - `File`: Prompts for entry of a file name, and then writes the list of bindings for the current mode to the specified file. It creates a file if necessary.
  - `Printer`: Prints the list of bindings to a printer, offering the local printer as a default.
  - `Help-help`: Provides descriptions of the commands under `All-commands`.
  - `Leave-help`: Exits the On-line Help facility and returns to the most current location prior to requesting help.
  - `Quit-list-keys`: Exits the current level of help and returns to the previous level, which is the menu for `Key-bindings`.
- `Help-help`: Provides descriptions of the commands under `Key-bindings`.
- `Leave-help`: Exits the help facility and returns to the most current item prior to requesting help.
- `Quit-key-help` Exits the current level of help and returns to the the menu for `Help`

Direct Help Via NMODE Commands

This sections discusses the direct access routine for getting help in some detail.

The third major way to get help is to execute a command that provides direct access to on-line documentation items. This routine is very efficient when you know what help you want and know which command to execute.

The **M-X** command names are the same as the command names. You can find the alternative bindings by executing **Help**, **Key bindings**, **Description** and typing the command name.

Commands That Duplicate Browser Commands

In essence, the following commands provide direct access to the same help that you can get by using the menu driven routine for getting help in a browser.

M-X Apropos	Prompts for entry of a string (e.g. `file` or `lisp`) and then displays a list of commands that relate to the string. This command is also available as **M-X Help Apropos**.
M-X Help Bindings	Works as if you had executed `Help` and then `Key-bindings`.
M-X Show Key Binding	Prompts for entry of a key sequence or **M-X** command name and then redisplays the entry and the corresponding, bound command function or indicates that the entry is not bound. **M-/** is an alternative key binding.
M-X Help Explain	Prompts for entry to a string to be explained and then displays items related to the string, together with an explanation of what the string represents. Alternatively, you can execute **M-X Explain**.

See the *NMODE Command Reference* manual if you need more information about these commands.

Help Tools

Several NMODE commands let you get information that answers particular questions; for example: What is the current location of the cursor? While these commands are not necessarily part of the help facility, they are discussed here because they provide information you can get "on-line" by executing a command.

M-X Count Occurrences	This command determines and briefly displays the number of occurrences of a specified string after the point in a buffer.
M-X Find Item	This command, which works with code indexes, lets you locate source code for a specified or default function that is contained somewhere in the files index. **M-.** is an alternative binding.
M-X Show Fill Values	This command displays the current values of system variables related to the left margin, indentation, right margin, and fill prefix.
M-X What Cursor Position	This command displays information about the cursor's current location. **C-X =** is an alternative binding.
M-X Show Function Bindings	This command prompts for the name of a command function and then shows all the current bindings for that function.

Help During Input Mode

It is often desirable to let the system complete a long command, buffer or file name for you. If you type enough letters to uniquely identify a name, pressing the space-bar will cause the name to be completed. If however, the system "beeps", then either more than one name matches the letters you have typed or there is no name that matches the letters you have typed.

If multiple names exist, you can have the system present you with a list of all matching names by typing a question mark. This places you in Prompt Broswing Mode which is a special extension of Input Mode. While in Prompt Browsing Mode, you can use search, filtering and grouping commands (discussed in future chapters) to see which names you have available to complete the pending input operation. Browsing an item in the list completes the pending input operation with the selected name. The Quit command in Prompt Browsing Mode returns you to Input Mode (where you typed the question mark). See the section on **M-X** Command name in the *Basic Use of NMODE* chapter for more information.

A Short Tour of the Help Command

Try the following to become familiar with the `Help` command.

1. Go to `NMODE Root` by typing **M-X** Nmode Root.
2. Press $\boxed{\text{H}}$ for help. A help menu will appear:

```
Where-am-I Explain Key-bindings Documentation Tutor Help-help Quit-help
```

Typing $\boxed{\text{W}}$ for Where-am-I will cause a short description of the current mode or application to appear on the screen. What gets displayed depends upon where you were when you asked for help. You can now type $\boxed{\text{Q}}$ to quit or $\boxed{\text{L}}$ to leave the help system.

The distinction between "leaving" and "quitting" the help system becomes important when you go "deeper" into the help system. For example, typing $\boxed{\text{H}}$ for help and then another $\boxed{\text{H}}$ for help-help causes information about how to use the help facility to appear. If you now "quit", you will return to the previous help menu. If you "leave" you will completely exit the help system.

Help with Key Bindings

Try the following:

1. From `NMODE Root`, press `H` for `Help`.
2. Press `K` for `Key-bindings`.
3. Press `D` for `Description`.
4. Now type a key sequence such as `CTRL`-`X` `CTRL`-`S`. A message will be displayed that shows the key sequence and its current command binding. In a moment, the screen will display information about the command and any other key bindings.
5. Press `L` to `Leave` the help system.

This ends the short tour. The other commands work in about the same way as the two commands you toured. Experiment with the help facility and use it according to your needs and preferences.

Chapter 5
Introduction to Browsers

Introduction

This chapter contains fundamental information about browsers. The next several paragraphs provide an overview that lets you know what a browser is and how a browser helps you use the NMODE environment. The remainder of the chapter extends information contained in previous chapters by providing detailed descriptions of how browsers work.

To get started, the following information items provide some background for defining a browser.

- You already know that NMODE often displays a list of items and that you can point to one of the items and browse into it.
- You also know that the items in the list are write-protected, which generally means that you cannot insert typed characters into the items.
- When you see a list of items, you know that, besides executing NMODE commands, you can execute special commands that appear in a menu in the message area.

The interface and mode in the NMODE user environment that provides a list of items, a non-character-insert (write-protected) state, and a menu of special commands is called a **browser**. Think of a browser as a tool, having a browsing mode, that lets you manipulate a set of items of a certain type. For example, the browser for `Directories` lets you manipulate files in directories, while the browser for `Buffers` lets you manipulate current editing buffers.

Besides letting you manipulate items, the browsers provide a very convenient way to move around the NMODE environment. To some extent, your ability to use NMODE is determined by your skill in using the assorted browsers.

NMODE automatically provides browsers for some facilities. You need to load other browsers. Each chapter that discusses a system facility mentions any procedure required to use the associated browser.

As you work through the rest of this chapter, and succeeding chapters, just remember that invoking any browser:

- Displays a "title" above the list of items and an entry in the mode line that tells you which browser you are in.
- Displays a list of write-protected items. The type of item depends on the browser. For example, when you see `Browser (Buffers)` in the mode line, the items are buffers such as `MAIN`, `OUTPUT`, or `MYCODE`.
- Provides a menu, in the message area, of special commands that you execute by pressing the capitalized letter in the command's name. These commands are in addition to the usual NMODE commands provided by the browser mode.

Using Browsers

The interface for a browser is consistent. Take a moment to examine this screen display.

```
        Title:   NMODE Root

            /       Buffers
           /        Directories
      Items:        Documentation
           \        User Options
            \       Additional Facilities


    Mode line:      @ Browser (NMODE Root)

 Message area:      Help  Browse  Group  Filter  Create  Options  Kill  Quit
```

Certain entries in the display indicate your current level and location in the NMODE environment. Others indicate items that can be manipulated and commands that can be used. Here is some information about each major part of the display.

- The message area contains special commands. Each browser has its own set of special commands. You can execute these commands in addition to any available NMODE commands.
- The title in the display, `NMODE Root`, indicates which which browsers you are currently using within NMODE. In the above screen, `NMode Root` means that you are at the **top** or **root** level of NMODE. Additionally, the display title ofter indicates the type of the items in the browser's list. For example, if the title is `Buffers`, you know that the items in its list are editing buffers. If the title is something like `Directory: /users/john/new-code/`, the items are files and directories in the indicated directory.

- In the above display, the list contains facilities called `Buffers`, `Directories`, and `Documentation`, among others. These facilities are the items that you can point to and browse or manipulate. The items of a browser can be facilities, directories, files, buffers, code indexes, source code lines, and so on. If you begin at the top level and browse to successively lower levels, you will move from more general facilities to more specific types of items.

  For example:

  a. Browse `Directories`, which is a top-level facility;

  b. Browse an active directory such as `$HOME/`, which is an intermediate-level item;

  c. Browse a certain file such as `.nmoderc`, which is a lower-level item; and

  d. Edit a specific portion of source code in the file; for example,

     `(nmode:nmode_invert_video).`

     which is a very specific piece of code.

With this information in mind, the next few subsections contain information that can help you move around in the browsers provided by the NMODE environment.

Accessing Items at Various Levels

In using browsers, it is helpful to think about **how** to access some items at the same time that you think about the items themselves and where they are located. For example, suppose you want information related to the Find File command. This information "sits" at a low level within the NMODE environment. Starting at `NMODE Root` (the top level), you could:

1. Point to and browse `Documentation`, a top-level facility.

2. Point to and browse `NMODE Commands`, a particular online document at the second level of NMODE.

3. Examine the long list of commands, and seeing that the Find File command does not appear on the first paneful of commands, scroll the display until you can point to and browse `Find File`. Note that the command is a particular online item at the third level of NMODE.

4. At the fourth level, you see assorted information about the Find File command. For example, you see the name of the function that executes the command and the name of the extended command. Notice that, when you get down to this low level, the mode line displays `Text Browsing` to indicate that you are in a mode that lets you read basic online information. Notice also that at this level, the `Browse` command is no longer available in the menu because you have reached a sufficiently

low level that further browsing is not possible. That is, you are at the lowest level of NMODE along the "path" that you browsed.

This procedure for using a browser was introduced in earlier chapters as the **point and browse** routine. Most people employ this routine within browsers until they become sufficiently familiar with the overall environment to use more efficient routines. For example, a **direct access** routine (M-X, C-X, or softkey command, for example) provides a much more efficient way to get to an exact item, regardless of level, but you need to know the name of the item and the command that accesses it.

Working With Items in a Browser

Every browser contains a highlighted region that indicates the current (selected or pointed to) item in the list. Moving the highlight to a different item selects that item. The highlighted region also contains the cursor.

To move the highlight down, and thus select a lower item in the list, use any of these: the Down-arrow key [▼], or the **Spacebar**, or the [Return] key.

To move the highlight up, and thus select a higher item in the list, use any of these: the Up-arrow key [▲], or [Back space].

You can also use the NMODE commands, **C-N** and **C-P** to move the highlight down and up, respectively.

You can skip several items by prefixing your command with a universal argument. For example, **C-4** followed by [▼] will move the highlight down four items. The [Next] and [Prev] keys will also skip a screenful of items at a time.

If you have window management software and a mouse, move the pointer onto a desired item and click the left button. The highlight moves to that item. A second click will browse the item.

The Menu of Commands

The commands in the menu for a browser differ from "regular" NMODE commands in the following ways.

- No prefix-character is required to execute a command because browsers are **not** invoked in a character-insert state as is the Emacs editing mode. Just press the capitalized letter in the command's name to execute a command. With a mouse, put the pointer on the command and click the left button.
- Each browser has its own menu of commands. The available commands relate to the functionality of the browser.

Some commands, such as `filter`, `group` and `help` prompt you with a second level menu of commands. Second level menus replace the original menu of commands with the name of the command you just entered, such as `'GROUP:'`. the input line then contains the menu of commands available. Execute a command using the capitalized letter of the command or the left mouse (locator device) button. Use the `quit` command in a second level menu to return to the original command menu without executing a command.

Many menu commands can use command arguments to execute the command on consecutive browser items. For example, the `item` secondary command under the `filter` command can be executed on the next 4 items by entering `C-4 FI`.

Some commands such as `Quit`, `Help`, `Browse`, `Group`, and `Filter` appear in the menus of most browsers. A browser may have commands that are specific to that type of browser, such as `Add-files` in the index browser.

The next section discusses some commands that are common to most browsers. Other commands are discussed in subsequent chapters that discuss browsers. For quick reference to commands, see *Appendix B: Browser Menus*, which is a lift-out card.

Common Browser Commands

While each browser has its own menu of commands, the following commands appear in the menus of most browsers. Descriptions of commands related to a particular facility browser appear later in the chapter for that browser. The next several subsections describe the common commands.

Browse

Executing `Browse` usually "goes into" the current item in some sense. For example, browsing an active directory in `Directories` invokes the browser for the active directory. In other cases, browsing an item accesses the item so you can view or edit it. For example, browsing an item in a code index causes the corresponding source code file to be displayed in a buffer, ready for viewing or editing.

Create

Many browsers provide this command. Executing `Create` provides a menu of potential items that can be created. In the case of buffers, the menu looks like this:

```
Buffer  File  Other  Quit-create
```

Executing `Quit-create` exits the create menu and redisplays the top level menu for the browser. Selecting `Buffer` or `File` prompts for a buffer or file name and then browses the newly created buffer or file. In a different browser, you might see options for different types of items, and they will create and browse objects of the corresponding type.

Executing `Other` invokes a create browser. The mode line displays:

```
@ Browser (Create)
```

to indicate that you are in a create minor mode within a major browser mode. Potential items that can be created are presented in a list. For example,

```
Activate Directory
Buffer
Code Index
Compilation Error Index
Directory
File
```

and the menu of commands includes:

```
    Help  Browse  Group  Filter  Create-item  Quit
```

By putting the highlight on one of the items and typing C to Create-item, NMODE will create and browse an item of the specified type. Pressing Q to Quit, will return you to the browser in which you executed the `create-other`.

Note that creating a file or directory with the `create` command will actually create the file or directory on you disc and then browse you into a buffer or directory browser. This differs from the `find-file` or `visit-file` NMODE commands which do not create a file on your disc until you give a `save-file` or `write-file` command.

Group

The Group command allows you to select several items for a particular operation. For example, you could group all files in a directory that were created on the same date and then copy them with one command. An item is grouped when it is preceded by a >.

Executing `Group` provides a menu of second-level commands, which let you place items in a group. Other commands such as `Filter` use grouped items. When the multiple selection option is turned on, some commands directed to one item in a group will act on every item in the group. Examples of such commands include: `Move`, `Copy`, and `Print`. Here are the second level commands:

`Item`	This command includes a previously ungrouped item as part of the group. A > is then displayed in front of the item.
`Exclude`	This command removes an item from the group, removing the >.
`Matching`	This command prompts for entry of a string and groups all the items that contain the string in their display lines. This command does not exclude previously grouped items; it just groups more items.
`Differing`	This command prompts for entry of a string and groups all the items that do not contain the string in their display lines. Like `matching`, this command does not exclude previously grouped items.
`Clear`	This command removes all items from the multiple selection group.
`All`	This command includes all items in the multiple selection group.
`Reverse-all`	This command inverts the grouping of all items in the browser. Every item that was grouped becomes excluded and vice-versa.
`Quit-group`	This command aborts the command and returns to the current browser.

A command argument such as C-3 can be used before a group-item or group-exclude command to modify several consecutive items.

The Multiple Selection Option and Grouping

This option determines whether a browser command will affect a single item or a group of items.

When multiple selection is ON and the current item is grouped, certain browser commands act on all grouped items, not just on the current (highlighted) item. When multiple selection is OFF or the current item is not grouped, the command only acts on the current item.

Filtered items are never included in the multiple selection group, even if an item was grouped before it was filtered.

For example, here is a section of a directory browser's list.

```
   Directory:  $HOME/   (/users/joe/)
   Last Read:  1-Jun-86 09:00:34      (Multiple-selection is ON)

   File Name        Type           Size       Write Date       Permission Owner

   .cshrc          File            1878  7-May-86 08:24:51 -rwxr-xr-x root
 > ug01.text       Document       25931 11-May-86 10:57:52 -rw-r--r-- joe
   ug01intro       File           32303  7-May-86 08:24:52 -r--r--r-- moe
 > ug02.text       Document       86188  2-Apr-86 12:44:03 -rw-r--r-- joe
   ug02screen      File           93595  7-May-86 08:24:54 -r--r--r-- moe
 > ug03.text       Document       15270 31-May-86 18:00:03 -rw-r--r-- joe
   ug03help.text   Document       20521  7-May-86 08:24:54 -r--r--r-- moe
```

Notice that the value of the multiple selection option is ON in this example and three files have been grouped. If the highlight is placed on any of the grouped items, some browser commands (such as kill and print) affect all of the grouped items.

You may change the setting of the multiple selection option in NMODE General User Options.

Grouping commands operate only on displayed (un-filtered) items.

Filter (Hide)

The Filter command allows you to "prune" a long list of items to a more manageable list. It is a handy way to select similar items or create a list of items to be manipulated (i.e. printed, purged, etc.).

Executing `Filter` provides the following menu of second-level commands that let you filter items.

```
Item  Grouped  Non-grouped  Matching  Keep-matching  Undo  Quit-filter
```

Except for `Quit-filter`, the second-level commands imply the criteria for filtering.

`Item`	This command filters (hides) the current item, removing it from the displayed list in a browser, but the item is not destroyed or lost. A command argument such as **C-4** can be used before executing `filter-item` to hide several consecutive items.
`Grouped`	This command hides all grouped items; the items preceded by a >. Prior to executing this command, you need to use the Group command to designate the items you want to include in a group.
`Non-grouped`	This command hides all items in a list that are **not** grouped. If no items were previously grouped, the command hides all the items, which can be a bit shocking.
`Matching`	This command prompts for entry of a string and then **hides** all items that contain the string some where in their display line. For example, you can hide all the files owned by root in a directory browser.
`Keep-matching`	This command prompts for entry of a string and then **keeps** all items that contain the string somewhere in their display line.

`Undo` This command "undoes" (reverses) the effect of the most recent second level command under `Filter`, except `Undo` and `Quit`. For example, if you executed `Grouped` to hide a group of marked items, executing `Undo` returns the items in the group to the display.

When a filter is applied, the mode line will contain `<n FILTERS>` where *n* is the number of filter operations applied. `Undo` will reverse the action of only the most recent filter and reduce the `<n FILTER>` count by one. Consecutive `filter-item` commands will be compressed into a single filter.

Since more than one filter can be applied to make a list, more than one Undo may be needed to return to the original list.

An Example of Filter

The following example of `Filter` illustrates a use of successive filters to produce a manageable list of items. This example provides a demonstration of filtering the documentation in `NMODE Commands`. Execute these commands:

1. **M-X nmode root**;
2. Move the highlight to `Documentation` and execute `Browse`;
3. Move the highlight to `NMODE Commands` and execute `Browse`.

This displays a long list of items where each item is the name of a command (on the left) and a description of the command (on the right); for example, the title and top item look something like this:

```
Documentation:  NMODE Commands

Abort Edit Template  aborts edit of a language template.
```

Now for the tour:

1. Execute `Filter`. Then execute `Keep-Matching` and enter the word: `file` as the string. This will reduce the list considerably, displaying only commands that contain the term, file, in their display text, not necessarily in the command name. Notice the mode line, which indicates that one filter is applied.
2. Now, examine the abbreviated list. Use the cursor keys and the Filter Item command to hide several command items. Notice the mode line, which shows that two filters are applied.
3. Now, execute `Filter` and then execute `Undo`. Notice that you get back the list that had one filter applied. Execute `Undo` again and notice that you get back the entire list of commands.

Help

Executing `Help` displays a menu of commands that provide access to NMODE's Help facilities. The Help commands include:

```
Where-am-I Explain Key-bindings  Documentation Tutor  Help-help  Quit-help.
```

The use of these commands was discussed earlier in the chapter called "Getting Online Help".

Kill

Executing `Kill` irreversibly removes the selected (highlighted) item from the system. If a kill would remove a file or unsaved buffer, the command prompts for confirmation. Use this command with caution and think about what will happen before you press $\boxed{K}$.

Many browser items in NMODE (such as the MAIN and OUTPUT Buffers) are protected and cannot be distroyed with the kill command.

Options

Options allow you to customize the NMODE environment to your liking. For example, the general environment options allow you to change the current-window indicator, toggle the multiple-selection option, and set other variables in the system. The list of options depends upon your location in the environment. Directory options will appear if you are in a directory browser, code index options appear if you are in a code index.

To see all options in use, you can go to `NMODE Root` and browse the item called: `User Options`.

The NMODE environment contains approximately 20 options browsers that are stored in *$LISP/config*. Options browsers are automatically loaded when the facility that they control is loaded. All browsers have the following format:

```
User Options:  <name of browser> (Values save/restore is enabled)
   Data from file: "$LISP/hi/customize/<filename.opt>"

option-1                     value of option-1
option-2                     value of option-2

option-n                     value of option-n


@ Browser (User Options) <name of browser>

Help  Browse  Modify  Group  Filter  Sort  Write  Restore-default  Quit
```

where `<name of browser>` is the name of the related options browser and `<filename.opt>` is the name of the file in *$LISP/config/* that provides the data for the options browser.

Most items either toggle between possible values or prompt you for a string. Each options browser allows you to save your modifications. The next time NMODE is invoked, during initialization it will restore your previously saved options.

See the *User Options* Chapter for more information on how to use and save options and for a description of each of the available options.

Quit

Executing `Quit` returns you to the previous location. For example, suppose you are at the `NMODE Root` and point to and browse the item: `Directories`. Executing `Quit` takes you back to the root.

If there was no previous location, NMODE will "beep" and display the message: `No previous location`. In this case, simply browse some item or go to `NMODE Root` or any other place by issuing a command.

NMODE keeps track of where you are and where you have been. You can always return "the way you came" by successive "Quits", or you can always "move forward" by issuing commands that take you to a new location.

Sometimes you will see a command `Quit-<string>`, where `<string>` is a modifier. For example, `Quit-filter` takes you back to the browser from which you executed `Filter`.

`Quit` does not necessarily move you up a tree, closer to Root. Rather it remembers where you have been and retraces you path. For example, from a buffer, `M-X Options` takes you to the list of user options. From here, `Quit` will take you back to your most previous location (the buffer you were editing), not the browser which contains User Options.

Each Pane maintains its own stack of locations you have visited. `Quit` simply pops the previous location off the current pane's stack. Thus, you can be in the same buffer or browser in several panes and `quit` in each pane will return you to a different location depending on how you got to each.

Quitting Text Buffers

When you leave a browser and enter a buffer, NMODE displays a different menu:

```
    C-M-L to QUIT,  C-? for HELP,  C-X R for Root
```

Here, instead of pressing Q to "quit" and return to the previous location, you must type a "Control-Meta-L" by pressing the CTRL-Extend char-L keys at the same time.

Other options include typing CTRL-? to invoke the Help facility, or CTRL-X R to "goto" `NMODE Root`, or typing any other NMODE command that invokes a browser (for example, **M-X Buffers** which invokes the buffers browser).

Quitting NMODE

If you want to quit NMODE entirely, you need to execute either **M-X exit nmode** or **C-X Z**. This will prompt you for confirmation and then completely exit the system, returning you to your HP-UX shell.

Trash

Executing `Trash` marks the current item for future deletion by displaying a `T` at the left end of the displayed item. If an item is already marked for deletion, `Trash` unmarks the item for deletion, removing the `T`. No items are actually deleted until you exit the browser with `Quit` or `C-M-L`. Leaving the browser or pane with any other command will not cause the trashed items to be deleted.

Upon quitting the browser, all items marked with a `T` will be deleted. If there is an unsaved buffer or if you are "trashing" a file, you will be prompted for confirmation. Once deleted, a file cannot be recovered.

Other NMODE Commands Available

In addition to the commands listed in a browser's menu of commands, many NMODE commands are available while in a browser. In general, all EMACS mode commands that do not modify a buffer are available, so you can search a browser for a specified string, copy regions of the browser to a buffer, etc. The arrow keys are available for scrolling a browser vertically and horizontally.

Use `Help-key-bindings All Commands` to get a complete list of the commands available in a particular browser.

Summary

You now know the commands that are basic to all browsers. Other commands that are unique to a particular type of browser are documented in subsequent chapters.

Chapter 6
NMODE Root

Introduction

This chapter explains how to use the browser for `NMODE Root`, whose essential purpose is to provide access to the major facilities that let you do productive work. Think of `NMODE Root` as containing the dynamic "parts" of the NMODE environment.

On invoking NMODE Root, you see the following screen, which has been generally described in previous chapters. The list of facilities can be a bit different since what you see depends on your initialization files and whether you loaded any facilities during the current NMODE work-session.

```
NMODE Root

  Buffers
  Directories
  Documentation
  User Options
  Additional Facilities



  @ Browser (NMODE Root)

Help  Browse  Group  Filter  Create  Options  Kill Quit
```

This is the top-level of the NMODE hierarchy. From here you have easy access to all of NMODE's major facilities. You might want to consider `NMODE Root` as a "safe haven" to which you can return if you become disoriented during your initial use of NMODE. If you get stuck for any reason while you are learning how to use NMODE, execute **M-X Nmode Root** to invoke this browser.

The current user environment can be expanded by loading additional facilities. The current environment is seldom static. For example, by using the procedures described later in the chapter called "Code Indexes", you can add the facility named `Code Indexes`.

Like all browsers, the message area contains a menu of commands that can be executed at the top level. There is more about these commands later.

Invoking the NMODE Root

When booting NMODE is complete you are placed at the NMODE Root. In addition, any of the following procedures invokes the NMODE Root.

- Execute **M-X Nmode Root** or **C-X R** from any location or level in NMODE to exit the current browser or buffer and go to the top level.
- Press the softkey for the label called **Nmode Root**. If you do not see **Nmode Root** among the softkey labels at the bottom of the screen, you might need to cycle through the available sets of softkeys, which are described in Appendix C, "Softkeys and Softkey Labels".
- If you have an operational mouse (locator device), position the pointer in a gray area within a Nmode pane, click the right mouse button, point to and click `Nmode general >>` (either button), then point to and click `Places >>` (either button), and then point to and click `Nmode Root` (either button).

Using NMODE Root

This section answers the question: What does the NMODE Root do for me?

The top level browser is often your "point of departure" when you access a certain facility, and it is the level to which you often return when you finish a particular task. The major functionality provided by `NMODE Root` is that it lets you easily access any part of the NMODE environment.

Most of the facilities available at `NMODE Root` are either already loaded into the system or will be loaded when you browse them. Exactly what is initially loaded depends upon your initialization files.

Accessing Items

To access an item in a certain facility, you typically employ one of the procedures discussed earlier. For quick review, the next few subsections are a review.

Point and Browse

With this technique, you can access items by moving through the NMODE hierarchy a step at a time. You point to a facility and browse it. Then you point to an item in the facility and browse it. This continues until you get to the item you want to manipulate. Hypothetically, to get to a file in a buffer, you might successively point to and browse:

- `Directories`, which is a facility;
- `/users/ellen/tools/`, which is an active directory (item) in the facility; and
- `print-tool.l`, which is a file (item), in an active directory, that you might edit to correct bugs.

Direct Access

You execute NMODE commands that directly access the items you wish to manipulate. For example, to go directly to a file in a buffer, you might execute the **Find File** command (**C-X C-F**), and on the prompt, you might enter `tools/print-tool.l` relative to the default path of `/users/ellen/`. Then, you could edit the file.

No one routine is necessarily best for getting to items, in particular facilities, that you want to use during a working session. Deciding on which routine you want to use to access some item depends mostly on your personal style and experience in using Nmode.

The NMODE Root Menu of Commands

Like all browsers, the NMODE Root browser provides a menu of commands.

```
Help   Browse   Group   Filter   Create   Kill   Quit
```

The section called "Common Browser Commands" in the earlier chapter called "Introduction to Browsers" described all of these commands. Most browsers provide these base level commands.

Loading Additional Facilities

When NMODE is shipped, the root contains buffers, directories, documentation, user options and additional facilities. This provides a basic set of tools. Other facilities can be loaded for more functionality. They are not standard to reduce the amount of memory needed to load NMODE.

These other facilities can be loaded in three ways:

1. In `make-nmode` you can uncomment the lines which refer to the facility. After you re-make NMODE, the facility is pre-loaded in the NMODE dump file.
2. In `.nmoderc` you can uncomment the lines which refer to the facility. The facility is loaded each time you call NMODE.
3. When you are in NMODE, you can load a facility by browsing into `Additional Facilities` and either **Load** or **Browse/execute** the facility you want.

   The **Load** command loads the facility (and any associated options files) but takes no further action. The **Browse/execute** command can then be used to run the facility. If you **Browse/execute** a facility that has not already been loaded, you will be asked to verify that you wish it to be loaded. This procedure must be done each time you enter NMODE.

The *Installation and Overview* manual describes how to load optional facilities in `make-nmode` and `.nmoderc`.

Additional Facilities

One of the items at NMODE Root is `Additional Facilities`. The items in `Additional Facilities` are all optional facilities. That is, they are not loaded by the standard NMODE.

If an item has been loaded, an L appears before it in the item list. When the item is loaded, you can browse it, by going to NMODE Root and browsing the Additional Facilities and then browsing the item. Or you can enter the facility directly through a series of key strokes. The methods for entering the facilities are described in the chapters which describe the facilities.

In general, when you load an additional facility, a new item appears at NMODE Root. For example, loading `File Search Index` and creating a search browser results in the item: `File Search Indexes` appearing at the root.

The Execution Monitor and Execution Stack Analyzer are described in the "Debugging Tools" Chapter of the *Lisp Programmer's Guide.*

The other facilities are described in this manual.

- Program Editing Support is described in the "Working with Lisp Code" and "Working with Other Code" chapters.
- The HP-UX Access facility is described in the "HP-UX Access Facility" chapter.
- The Code Index is described in the "Code Indexes" chapter.
- The Compilation Error Index is described in the "Error Indexes" chapter.
- The File Search Index is described in the "Search Indexes" chapter.

key itf

Chapter 7 The Buffers Facility

Introduction

This chapter describes the top-level facility for accessing buffers (workspaces for editing text). In addition, it describes buffers themselves. This is an important topic because all editing occurs in buffers.

Buffers

This section describes buffers and some commands for manipulating them. You can manipulate buffers directly (the technique described in this section) or through the Buffers facility described in the next major section of this chapter. If you're just starting, you may want to read just the description of buffers that follows, and then skip ahead to the section that describes the Buffers facility.

A buffer (or editing buffer) is an area for viewing and manipulating text. A buffer may contain the contents of an existing file, or it may contain text that has not yet been saved in a file; at times a buffer may not contain any text at all. The number of buffers and the maximum size of a buffer are limited by the size of available memory (heap).

Buffers only last until the end of the working session. When you edit a file, you are really only editing a buffer that had the file read into it. The changes you make in the buffer are not saved to the file until you do it explicitly with the Write File or Save File commands.

Since you can have several buffers in the system at one time and files are edited within buffers, you can edit several files at the same time. This allows sharing of data between the files (buffers).

Buffer Characteristics

Buffers have the following characteristics:

name	The identifier of a buffer. Case is not significant in the name of a buffer. You can change the name of a buffer with the Rename Buffer command. The name of a buffer appears in its mode line only if it does not have an associated file, or if the name of the buffer is different from the file name.
associated file	This is an optional characteristic that determines the behavior of certain commands. For instance, the **Save File** command **(C-X C-S)** writes out a buffer to its associated file. (If there is no associated file, you are prompted for the file name, and that becomes the buffer's associated file.) The associated file name appears in a buffer's mode line.
modified flag	This indicates whether the buffer has been modified since it was last saved to a file. If a * appears at the right side of a buffer's mode line, the current contents of the buffer have **not** been saved.
modes	Each buffer has an associated major mode (Emacs) and zero or more minor modes (Lisp, Text, etc.). The modes of a buffer determine what editing commands are available and the behavior of certain commands.

Default Buffers

Two buffers are created automatically when NMODE is invoked: MAIN and OUTPUT. These buffers cannot be killed and are not initially associated with any files (although it is possible for you to do this). The buffer MAIN is conventionally used as a place for trying out small pieces of Lisp code. The OUTPUT buffer is where the system normally writes the return values of evaluated forms. To make some wasted space reclaimable by the garbage collector, you should occasionally delete lines in the OUTPUT buffer that you no longer need to see.

Mode Line Display for a Buffer

When you are in a buffer, the mode line looks something like this:

```
I/O @ Emacs (Lisp) [FOO] /users/kathy/code-red.l  {USER} -45%- *
```

The following items describe each part.

- The @, a string which is the default value of nmode::nmode-selected-window-symbol, indicates that the buffer is current (selected).
- The major mode is Emacs.
- The minor mode(s) appear in parenthesis, Lisp in this case. There may be more than one minor mode.
- The item in brackets, FOO, is the buffer name. This is displayed when there is no associated file, or the buffer name does not match the associated file name. The next item is the associated file name.
- The item in braces, USER, is the current package. This appears only for buffers with Lisp minor mode.
- The percentage indicates the position of the cursor in the buffer relative to the top of the buffer. In this case, the cursor is located 45% of the "distance" from the top of the buffer.
- The asterisk is the buffer modified flag, which indicates that the contents of the buffer have been changed since they were last saved to a file.

In many cases, the mode line does not display all these things, but in any case, the mode line indicates the current state of a buffer.

Buffer Modes

The following information items indicate how a buffer can be configured.

- An editing buffer defaults to Emacs mode, which means you have available the basic editing commands.

- Available minor modes in Emacs mode include: Text, Lisp, Pascal, C, Fortran, Auto Fill, and HP-UX. To invoke a minor mode, execute **M-X** *string* `mode`, where *string* is one of the minor modes. Note that you would enter just one of the strings inside the brackets. Each minor mode provides additional functionality that helps you edit text or code.
- You can enable or disable automatic filling by executing **M-X auto fill mode.** The command acts as a toggle. Alternately, you can press the **Auto Fill On** softkey, or use the popup menu item: `Auto Fill Toggle`.
- A special shell buffer is available for interacting with HP-UX if you loaded the HP-UX System Access Facility as described in the "HP-UX Access" chapter of this manual.

Manipulating Buffers

The Buffers facility discussed later in this chapter provides a high-level interface for creating, selecting, and destroying buffers. There are also direct means of dealing with buffers.

Creating and/or Selecting a Buffer

You can select an existing buffer or create a new buffer from anywhere in the NMODE environment by executing the **Select Buffer** command (**C-X B**). The command prompts for a buffer name, and then brings up the specified buffer. If a buffer with the specified name does not exist, an empty buffer is created and entered. The new buffer has no associated file. The new buffer defaults to Emacs mode and its minor mode is taken from the value of the special variable `nmode:nmode-default-language-mode`, which defaults to `nmode:lisp-language-mode`, but can be changed in your `.nmoderc` initialization file.

Exiting a Buffer

When you are in a buffer, there are no command line commands like `Quit`; typing a single key inserts the corresponding character into the buffer. To exit a buffer, use the **Select Previous Buffer** command (**C-M-L**). This will take you to your previous location in the current pane. Note that the name of this command is slightly misleading, since your previous location may not have been a buffer.

Associating a File with a Buffer

There are several ways to associate a file with a buffer. The chapter "Working With Text" discusses these in a more appropriate context, but they are mentioned here for completeness.

- Use the **Find File** command (**C-X C-F**). This will prompt for the name of a file, and create and select a buffer associated with that file. If the file exists, the contents of the file will be in the buffer.

- When in a buffer, execute the **Visit File** command (**C-X C-V**) which loads a specified file into the current buffer, overwriting the buffer's old contents.

- When in a buffer, execute the **Write File** command (**C-X C-W**), or the **Save File** command (**C-X C-S**) which write the contents of the buffer to the named file, and then associate that file name with the buffer.

- Writing to a file that is associated with another buffer removes the association file from the other buffer (but does not modify the buffer). This helps prevent two very different buffers from being associated with the same file.

Killing and Deleting Buffers

There are several ways to get rid of unwanted buffers. Note that these have no effect on the file (if any) associated with the killed buffer. However, remember that you may have made changes to the buffer that have not yet been written to the associated file. If you kill the buffer, those changes are gone for good.

- Execute the **Kill Buffer** command (**C-X K**) and enter the name of a buffer to be killed. The command prompts for confirmation if the contents were modified. If the current buffer's name is an associated file's name, due to previous execution of **Find File**, executing **C-X K** and hitting [Return] with no entry kills that buffer but not the file itself.
- In a buffer or the `Buffers` facility, execute **M-X kill some buffers**. This command displays the name of each buffer in turn, prompting each time for you to enter `Y`, `N`, `V`, or [ESC] to kill, not kill, view, or abort the command respectively. The `V` provides a recursive editing level in which you can move around in the buffer, but not change it.

Renaming a Buffer

To rename an existing buffer, select the buffer and execute **M-X rename buffer**. Enter the new buffer name and press [Return]. If the new name is the null string (no typed characters), an existing default string or the associated file name is used as the new buffer name. If you enter the name of a different, but existing buffer, a message is given indicating the name is in use and the current buffer name does not change.

The Buffers Facility

The buffers facility groups all existing buffers into a single location from which they can be easily accessed. In addition, it supplies command line commands for manipulating buffers.

From NMODE Root, if you point to and browse the line called Buffers, the screen display looks something like this:

```
Buffers

  Buffer Name                 Size   File Name

  MAIN                          55
* OUTPUT                       115
  TEXT                         265
  UG7.L                        283   /users/joe/ug7.l



I/O @ Browser (Buffers)

Help Browse Group Filter Sort Create Utility Options Kill Trash Write Quit
```

Notice that the title in the pane is: Buffers and the mode line displays:

```
@ Browser (Buffers)
```

to indicate that you are in the browser for Buffers. Also, notice that the browser provides numerous commands that let you manipulate buffers. The commands are discussed later.

The browser keeps track of buffers' names, sizes, and associated files. Unlike some browsers, the browser for Buffers has no related options browser that lets you control what is displayed.

Selecting the Buffers Facility

Use any of the following methods.

- **From NMODE Root**: Point to `Buffers` and execute `Browse`.
- Execute the **List Buffers** command, **C-X C-B** or **M-X list buffers**.
- If you have a locator device (mouse), position the pointer in the "desktop" area (outside any windows) and click the right button to get the popup menu called `NMODE GENERAL`. Then, point to `Places >>`, click either button, and point to and click `Buffers` (either button).

The Item-line for Buffers

In the browser for `Buffers`, the item for a particular buffer might look something like this:

```
 * OBJECTIVES           1225  /users/spanky/objectives
```

The buffer name, `OBJECTIVES`, and its size in lines, `1225`, are always displayed. The left end of the item can show several status flags:

*	Indicates that the buffer's contents have been altered since they were last saved to a file.
>	Indicates that a buffer is grouped (placed in the multiple selection group).
T	Indicates that an item is "trashed". That is, it will be killed when you exit the browser with `Quit` or **C-M-L**.
S	Indicates that the buffer is an HP-UX Shell buffer (described in the chapter "HP-UX Access").

Special Commands for Buffers

Upon going into the `Buffers` facility, you see the following menu of commands:

```
Help Browse Group Filter Sort Create Utility Options Kill Trash Write Quit
```

The commands named `Help`, `Browse`, `Group`, `Filter`, `Kill`, `Trash`, and `Quit` work as described earlier in the "Introduction to Browsers" chapter of this manual.

The next several subsections describe the commands named `Create`, `Sort`, `Utility`, `Options`, and `Write`.

The Create Command (Buffers)

Executing `Create` while in the `Buffers` facility provides the following menu of subcommands:

```
     Buffer   File   Other   Quit-create
```

These second level commands work as follows:

`Buffer`	Executing this command prompts you for a name and then creates and enters a new buffer with that name.
`File`	Executing this command prompts you for a name, creates a file with that name, and enters a new empty buffer associated with the new file.
`Other`	Executing `Other` provides a browsable list of other items you can create, such as directories.
`Quit-Create`	Executing `Quit-create` returns to the browser for `Buffers`.

The Sort Command (Buffers)

Executing `Sort` calls a menu of subcommands that let you sort existing buffers according to:

```
Buffer-name   Size   File-name   Modified   Reverse   Quit-sort.
```

These categories work as follows:

`Buffer-name`	Sorts items alphabetically by their buffer name.
`Size`	Sorts from smallest to greatest buffer size.
`File-name`	Sorts items alphabetically by associated file name. Buffers without an associated file are considered to be "before" all other buffers. Note that the full pathname of an associated files is used, not just the basename.
`Modified`	Moves buffers marked with an * to the top of the list.
`Reverse`	Prompts you for attribute with which items will be sorted in reverse order.

The Write Command (Buffers)

Executing `Write` prompts for the name of the file to which the highlighted buffer should be written. If the buffer has an associated file, that file is given as the default, so you can just type $\boxed{\text{Return}}$ to save a buffer to its associated file. After writing to a file, that file becomes the buffer's associated file.

The Utility Command (Buffers)

Executing `Utility` provides a menu of subcommands:

```
Not-modified Print-buffer Rename-buffer Set-filename File-revert Quit-utility
```

These second level commands work as follows. Where "buffer(s)" appears, it means that those commands work with multiple selection groups or command arguments.

`Not-modified`	removes a displayed * and lets you treat the current buffer(s) as though it was not modified.
`Print-buffer`	prompts for entry of a print device, offering `lp` as a default to print the current buffer(s).
`Rename-buffer`	prompts for entry of a buffer name and changes the buffer's name to the new entry.
`Set-filename`	prompts for entry of an associated file name, sets the associated file to the entry, but does not read the file. To remove the associated file name, just press Return.
`File-revert`	prompts for entry of `Y` or `N`. Entering `Y` replaces the current buffer(s) with the associated file from a disc, thereby letting you revert back to a previous copy of a file.

The Options Command (Buffers)

There is no options browser exclusively for the Buffers facility. Executing `Options` takes you directly to "Nmode General Users Options". You do not pass Go, you do not collect $200.

Updating the Buffers Facility

If the browser for the Buffers Facility is displayed in a pane other than the current one, it is updated to accurately show what buffers exist, but the size and modification fields are not updated until the browser's pane is selected.

Chapter 8
Directories Facility

Introduction

This chapter describes the following items.

- The active `Directories` facility.
- The Browser for a Directory.

Before looking at them, a short review of the HP-UX file system is in order.

The HP-UX File System

HP-UX uses a hierarchical file system. This simply means that each directory not only can contain files but can also contain other directories. The top-level directory that contains all other directories is named: / (slash) and is called the "root" directory. Thus, to access a file that exists inside a directory which exists inside yet another directory, you "append" the directories with a "/" to form a **pathname**. For instance, the file: `bugs` exists inside the directory: `joe` which exists inside the directory: `users`. The pathname would be written `/users/joe/bugs`.

Sometimes it is useful to shorten the pathnames. An HP-UX shell provides variables that can be set to pathnames. For example, the shell variable: `HOME` could be set to the value: `/users/joe`; then you could access the `bugs` file by using the path: `$HOME/bugs`.

Other common directory specifiers include the double-dot (`..`) which means the directory "above" the current directory, the single-dot (`.`) which means the current directory, and `$LISP` which means the directory that contains all of the Lisp files.

Active Directories

Starting at NMODE Root, the facility called Directories contains **Active Directories**. Active directories are directories that have been loaded into memory. They can be loaded in your NMODE initialization file or loaded upon request.

If you point to and browse the Directories facility, the screen might look something like this:

```
Active Directories

   $HOME/
   /
   /lisp/config/


 @ Browser (Directories)

Help  Browse  Group  Filter  Create  Options  Kill  Quit
```

Note that the mode line says "Directories". The commands such as Help, Group, Filter, and Quit were discussed in the section called "Common Browser Commands" in the chapter called "Introduction to Browsers".

Here are the remaining commands and a brief description of each.

Browse	Browsing an item in the Active Directories browser puts you in a Directory browser of the specified directory.
Create	The create command provides a submenu that lets you create an empty directory, or **activate** an existing directory. Use the Activate-directory command when you want to browse a directory without browsing each of its parent directories.
Options	This command invokes the NMODE General User Options options browser. (See the "Users Options" chapter of this manual for the description.)

`Kill` In the active directories browser, killing an active directory removes the item from the list but does not remove the directory itself. To return the directory to the list you will have to press $\boxed{\texttt{C}}$ to create then press $\boxed{\texttt{A}}$ to activate the directory. You will then be prompted for the name of the directory to activate.

Note that the previous commands are for the `Active Directories` browser and not for a `Directory` browser (which is explained next).

The Browser for a Directory

The browser for `Active Directories` shows all directories that have already been loaded into memory (either by the initialization file or by your request). The `Directory` browser shows all files and sub-directories that exist in a particular directory. The files can be source code (programs), compiled code, text, data, and so on. The directory browser "understands" most file types and knows which commands and operations apply to each type.

A Directory Browser Screen

When you invoke a directory browser (often by browsing an active directory) the screen display might look something like this:

```
Directory: /users/tom/ace-project/
Last Read: 19-May-86  09:31:39      (Multiple-selection is ON)

 File Name        Type         Size  Write Date          Permission  Owner

 ./             Directory      1024  30-Apr-86 13:23:48  drwxr-xr-x   joe
 ../            Directory      1024  12-Apr-86 14:03:31  drwxr-xr-x   joe
 zap-tool       File          20345  19-Apr-86 15:22:56  -rw-r--r--   joe
 zap-tool.txt   Document      37779  19-Mar-86 16:12:56  -rw-r--r--   tom
 zap-tool.l     Lisp Source   12479  20-Apr-86 08:42:17  -rw-r--r--   joe
 zap-tool.b     Lisp Code     15295  19-Apr-86 10:27:39  -rw-r--r--   joe
 zap-tool.c     C Source      32185  19-Apr-86 17:41:02  -rw-r--r--   root


  @ Browser (Directory) /users/tom/ace-project/

Help Browse Group Filter Sort Create Utility Options Type-specific Quit
```

As you can see, the Directory browser is like all browsers in that it has a title, a list of items (files or directories), a browser mode (write-protected state), and a menu of commands, but there are some powerful differences.

Notice that information is displayed with the title to describe the name of the directory, the last read time, and the state of the multiple-selection option. Notice also that, for each file, you get information about the name, type, size, write-date, and permissions. The display of these things can be customized by executing Options, which is explained later.

The value of the Multiple-selection option can be ON or OFF. If it is ON and you used the Group command to group a set of files, then commands such as Print, Kill, Move, and Copy act on all files in the group when the current item is in the group, not just on the selected file.

Notice that the items can be any type of file: text, document, Lisp source, Lisp code, C source, and so on. This allows you to construct directories related, for example, to a current project or an application you are developing.

Notice that a one-dot directory means the current directory. Browsing the one-dot directory will cause NMODE to re-read the directory. This is an easy way to update NMODE's copy of the directory if you changed something while you were in an HP-UX shell. NMODE automatically updates if you change something from within NMODE, but cannot know if something was changed by an HP-UX command given in another shell or by an HP-UX shell-buffer in NMODE.

A two-dot directory indicates the directory that contains the currently displayed directory. Browsing the two-dot directory takes you "up" the hierarchy.

Entries for files look something like this:

```
>T  mydocs      File         37948  14-Apr-86  10:02:03 -rw-r--r--  joe
 B* mydocs.doc  Document       345  23-May-86  11:12:13 -rw-rw-rw-  tom
 b  program.l   Lisp Source  12345  27-Apr-86  14:23:45 -rw-rw-rw-  guest
```

Again, be aware that the information in the line can vary, depending on the values of user options for the browser. The characters to the left of the file names are flags indicating the status of the file.

Flags

Each file can have one or more flags associated with it. The flags indicate the status of the file (and/or its associated buffer) as follows:

>	This flag indicates that the file has been included in a group by the Group command. When the multiple-selection option is "ON", all grouped files are affected by an operation (such as Print) instead of just the selected file. The flag is toggled by the Group command.
T	This flag indicates that the file has been "Trashed" and will be deleted when you leave the current directory with Quit or **C-M-L**. The flag is toggled by the Utility menu's Trash command.
b	The lowercase "b" flag indicates that a temporary buffer has been created by NMODE for the file. This usually means that the file has just been browsed but not modified. Once the buffer is modified, the buffer becomes "permanent" and the file is flagged with an uppercase "B". Each time you browse another file in the same directory, the previous temporary buffer is killed and a new temporary buffer is created.
B	The uppercase "B" flag indicates that a buffer has been created by NMODE for the file.
*	This flag indicates that the buffer (containing a copy of the file) has been modified since the last time it was read or saved. Usually, when you modify a file, the flags: `B*` appear next to the filename. After you save the file, the flag `B` remains, indicating that a buffer containing the file is still loaded in memory but that the contents exactly match the file saved on the disc.

When you browse a "flagless" file, a buffer is created for that file and the file is read into the buffer. However, if a buffer already exists for that file, you will enter the buffer without re-reading the file. Thus, if you were to use some HP-UX shell command to change the contents of a file that has a `B` flag, browsing the file would actually show you the contents of the buffer. You can use the **M-X Revert File** command to force the reading of the file into the buffer.

File Types and Naming Conventions

You can enhance NMODE's "power" in many cases by consistently appending appropriate suffixes to your file names. When you save a file with no suffix, NMODE does not treat the file in any special way; it is just a file. With a suffix, the NMODE provides functionalities that match the type of file. The types of available suffixes are presented here:

Type of File	Suffix
Common Lisp Source	.l
Compiled Lisp Code	.b
C Source	.c
Pascal Source	.p
Fortran Source	.f
Compiled C, Pascal, and Fortran	.o
Text File	.txt
Document File	.doc
User Options Data	.opt
Code Index Data	.cb
Search Index Data	.sb
Compilation Error Index Data	.eb

A directory browser understands most file types and knows which operations apply to each type. Thus, for example, when you browse a file of a certain type, NMODE does not just browse the file; it also invokes constructs related to the file such as the correct mode. A consistent use of suffixes that indicate file-type will help ensure that you will get into a "correct" state when you browse or otherwise manipulate a file.

Calling a Directory Browser

There are several ways to enter a directory browser.

Entering a Directory Browser from NMODE Root

Suppose you are at the top level of NMODE, either by invoking NMODE or executing **M-X Nmode Root**. To get into the browser for an active directory, you can point to `Directories` and execute `Browse`. Then you can either browse one of the active directories or activate another directory.

Entering a Directory Browser from Anywhere
When you are not at the top level of NMODE and you want to call a directory from any current location in the environment, you can do any of the following.

- Executing the **Edit Directory** command (**C-X D** or **M-X find directory**) prompts for entry of a directory name relative to the current pathname; for example,

  ```
  (relative to '/users/roger/')
  ```

  On entering a name, the command finds the specified directory and invokes the browser for that directory. If the specified directory does not exist, the command indicates in the message area that the directory cannot be found and aborts.

- To use a softkey, you can, if necessary, toggle the sets of available softkey labels until you see **Directories** and press the corresponding softkey. This invokes the `Directories` facility and you can then point to and browse into the active directory you want. Alternately, you can, if necessary, toggle the sets of available softkey labels until you see Find Direct. and press the corresponding softkey. This executes the **Edit Directory** command.

- If you have an operational locator device (mouse) and you are in a buffer, clicking the right button provides a popup menu. Point to and click `Nmode General >>` (either button), which provides another popup menu. Then, point to and click `Places >>` (either button), which provides yet another popup menu. Point to and click `Directories` (either button), which invokes the `Directories` facility. From here, you can browse into or activate a directory.

The Menu of Commands for a Directory

A browser for a directory provides the following menu of commands:

```
Help Browse Group Filter Sort Create Utility Options Type-specific Quit
```

The commands for `Help`, `Group`, `Filter`, `Create`, and `Quit` work according to descriptions given earlier in the section called "Common Browser Commands" in the chapter called "Introduction to Browsers".

Browse (Directory)

Executing `Browse` does one of these things:

- When the highlighted item is a directory, executing `Browse`: invokes a browser for that directory, and if necessary, loads (activates) the directory. This lets you examine subdirectories. For example, you can find a file by tracing its pathname step by step, browsing each level of its name.
- When the highlighted item is a file that can be edited, the Browse command simply enters the file and lets you examine or edit it.
- When the highlighted item is a file with a "known" suffix, such as any of the files ending in "`.opt`", browsing the file will invoke the appropriate browser.
- When the highlighted item is a Lisp Code file (`.b`) browsing the file will load the file.

Sort (Directory)

Executing `Sort` provides a menu of commands.

```
Filename  Grouped  Type  Size  Write-date  Reverse  Quit-sort
```

You can sort files in a directory according to the criterion implied by the subcommand.

`Filename`	Sorts files into an alphabetical list.
`Grouped`	Places grouped files at the top of the list.
`Type`	Sorts files according to their type; for example, `C-Source`, `Directories`, `File`, `Lisp Source`, etc.
`Size`	Sorts files according to file size, smallest to largest.
`Write-date`	Sorts files by write-date, earliest to most recent.
`Reverse`	Prompts for sort command to be applied to the directory but in reverse order.
`Quit`	Exits the Sort command, returning to the File Browser.

Sort will preserve the previous ordering within the newly sorted list when possible. For example, sorting by Size then sorting by Group will place the grouped files first and the grouped files will be ordered by size.

Utility (Directory)

Executing `Utilities` provides the following menu of commands:

```
Move Copy Rename Save Kill Trash Print Update File-permission Quit-utility
```

which work as follows:

`Move`	Prompts for the name of a destination directory or file name. If only a directory is given, the current filename will be used. The command moves the current item to the new location and removes it from the original location. You cannot move a directory.
`Copy`	Prompts for the new name, then copies the highlighted item to a specified directory or file.
`Rename`	Prompts for an existing name and a new name and then renames the file. You cannot rename a directory.

`Save`	Saves (writes) the highlighted file if it has been modified or indicates that no changes need to be written. Note that the Write File command (**M-X Write File** or **C-X C-W**) will write the Browser's displayed contents and not the file.
`Kill`	Permanently removes the file from the system.
`Trash`	Places a `T` before the item to indicate the item will be "trashed" (killed) when you exit the browser. Upon exiting the browser with Quit or **C-M-L**, you will be prompted for confirmation to kill the items. A second Trash command will toggle the indicator.
`Print`	Prints the highlighted file to your local printer. See the Print Buffer command in the "Working with Text" chapter of this manual for a description of printer commands. Note that **M-X Print Buffer** or **C-X C-P** prints the browser's display and not the currently highlighted file.
`Update`	Re-loads the current directory and updates the display. This is needed after the directory has been modified by some HP-UX command, Lisp process, or something other than NMODE. (This is the same operation as browsing the file ./.)
`File-permission`	Lets you alter the access permissions of the highlighted item. (See the next section.)
`Quit-utility`	Exits the Utility command and returns to the browser.

Note that the `Move`, `Copy`, `Save`, `Print`, and `Kill` commands will operate on all grouped items if the current item is grouped and the multiple-selection option is on.

Type-specific (Directory)

Executing `Type-specific` displays a menu of second-level commands under the top-level menu for the browser according to the type of the current file. Menu items depend upon the file type.

Lisp Source (.l)

`Browse-code`	Creates a Code Index for the current file.
`Edit`	Edits the current file (same as Browse).
`Interpret`	Evaluates `lisp:load` on the current file.
`File-Compile`	Compiles the current file to the specified destination file (`.b`) just like the **Compile File** command.

Lisp Code (.b)

`Load`	Evaluates `lisp:load` on the current file (same as Browse).
`Edit`	Read the current file into a buffer (use with caution)

C, Pascal, Fortran Source (.c .p .f)

`Browse-code`	Creates a Code Index for the current file.
`Edit`	Edits the Index Data file (same as browse).
`File-Compile`	Invokes the HP-UX compiler for the current file (like the **Compile-File** command).

User Options Data File (.opt)

`Browse-options`	Loads and enters the User Options (same as Browse).
`Edit`	Edits the Options Data file.

Code Index Data File (.cb)

`Load`	Loads the Code Index and creates the browser items (same as Browse).
`Register`	Loads the Code Index but does not create browser items.

Compilation Error Index Data File (.eb)

`Load`	Loads the Error Index and creates browser items (same as Browse).

Search Index Data File (.sb)

`Load`	Loads the Search Index and creates browser items (same as Browse).

Network Special File (files in /net)

`Connect`	Allows you to make a RFA (Remote File Access) connection to the current network special file.
`Disconnect`	Disconnects the RFA connection on the current file.
`Edit-network-special-file`	Reads the actual contents of the network-special-file into a buffer for editing. (Use with Caution.)

If the current file type has no subcommands, you briefly see the message:

```
No type-specific commands for this item.
```

Options (Directory)

Executing `Options` calls the user options for directories and displays a list of system variables that can be given values which control the operation of the Directory facility.

See the "User Options" chapter in this manual for more information on the operation of the Options facility.

```
  User Options:  Directory Options
    Currently saved in "$LISP/config/directory.opt"

 CONTROL OPTIONS:
   Multiple selection of Group (>) items      Yes
   Read headers to type files                 No

 FIELD DISPLAY OPTIONS:
   File Type                                  Yes
   File Size                                  Yes
   Write Date                                 Yes
   Read Date                                  No
     Time with Read/Write Date                Yes
   Access Permissions                         Yes
   File Owner                                 Yes
   File Group                                 No
```

User Options for Directories

The option for "`Multiple Selection of Group (>) items`" is a duplicate of the option in `NMODE General User Options`, so modifying the value in either browser will affect all browsers.

The option for "`Read headers to type files`" controls whether the Directory facility will open and read the "`Language`" field of the standard file-header (if it exists) of every file in the directory when the directory if first activated and when the Utility menu's Update command is issued.

If the options is the default, "NO", NMODE will use each file's suffix to determine the file's type. Although enabling this option allows NMODE to do a better job of determining the file type, reading and updating a directory will be slower.

By changing the field display options, you can control what fields are displayed in all browsers for directories.

File Access Permissions

Each file in the HP-UX file system has access permission indicators that show you who may access the file. The permissions are broken-down into three sets. The first set is for the owner of the file. The second set is for the "group" of people who work with the owner of the file. The third set is for all other users on the system.

```
OWNER   GROUP   OTHER

 ---     ---     ---
```

Within each of the three sets there are three indicators. These represent read-permission, write-permission, and execution-permission. A letter indicates the permission is allowed, a dash indicates it is not.

```
OWNER   GROUP   OTHER

 rwx     rwx     rwx
```

In addition to the nine permission indicators, a tenth indicator at the beginning of the list shows whether the file is a directory, a special file, or some other type of file.

```
TYPE      OWNER   GROUP   OTHER

  -        rwx     rwx     rwx
```

Putting it all together, here is a line from a Directory browser:

```
parts           Directory       1024  22-Apr-86 14:03:31   drwxr-x---    tom
```

The second to last field shows the permissions. The d appears since this "file" is a directory. The owner, tom, may read, write, or execute the file. (For directories, "execute" means to be able to go through the directory to a subdirectory.) People in the owner's group may read, execute, but not write; while all other users cannot read, write, or execute at all.

Now if you understand that, imagine that each set of three permissions were actually an octal digit. The r would be of weight 4, the w would be of weight 2, and the x would be of weight 1. Thus, specifying a permission of 644 would set the bits to be rw-r--r-- (and would appear on the screen as: drw-r--r-- since it is a directory).

In another example, setting permission of 777 would result in the permission: rwxrwxrwx. This would allow anyone to read, write, or execute the file.

If you wanted to keep the contents of a file a secret, readable only by yourself, you would specify: 400.

You can use these octal values to modify the permissions of a file with either the Utility menu's File-permission command from any Directory browser or by the **M-X File Permission** command from anywhere in NMODE.

Only the owner of a file may change its permissions.

Local Area Networks

Your HP-UX operating system may have the optional software product that supports a Local Area Network (LAN). If you do have LAN hardware and software, NMODE supports Remote File Access (RFA) on the LAN.

The NMODE commands are:

- **M-X Network Connect**
- **M-X Network Disconnect**

The **Network Connect** command activates the directory containing the network-special-files (usually /net but can be set in the `NMODE General Users Options` options browser). You are prompted for a network-special-file, your login name, and password. Defaults are provided. Once connected you can use any file access command as you would on your own system.

LAN and the Directory Facility

Browsing the /net directory provides connection status information on each of the network-special-files listed there. Browsing one of the files will attempt the connection sequence. If successful, the root of the remote file system will be browsed.

If you make the remote file access connection in an HP-UX shell before you boot NMODE, the /net directory will not show the connection. However, the files on the network can be accessed through commands such as **Find File**. If you browse a network-special-file that was connected before you entered NMODE, another connection is required.

There are type-specific commands for network-special-files to connect, disconnect or edit the current network-special-file.

Chapter 9
Working with Text

Introduction

This chapter describes how you can "work with text" within the NMODE user environment. In this general context, **working with text** means writing, editing, updating, and otherwise manipulating text that can be a program, document, report, and so on. The chapter contains basic information that is assumed in subsequent chapters about working with Lisp and other code.

NMODE provides an **EMACS-type editor** automatically whenever you browse into a buffer that has **Emacs major mode**. What you should know is that, when the mode line indicates you are in a buffer with **Emacs** mode, you are in the **EMACS-type editor.**

Organization

The organization of this chapter anticipates that you will typically do the following things in order.

1. Begin an editing session by establishing a suitable environment.
2. nUse commands and procedures to write or edit.
3. End the session by saving or printing your text.

Thus, the chapter is organized according to the following general topics:

- "Establishing An Editing Environment" describes accessing and manipulating buffers; invoking and utilizing modes; manipulating screens, windows, and panes; and setting up global values for margins and indenting.
- "Fundamental Text Editing" describes use of prefixed arguments; inserting and deleting characters; moving the cursor; and basic ways to manipulate characters, words, lines, sentences, and paragraphs.
- "Text Manipulation" describes more complex procedures and commands for manipulating text. The text can be anything from one character to an entire buffer or file. This includes transposing characters, words, lines, and regions; setting marks and establishing regions; killing and yanking units of text; and moving or inserting buffers and files.

- "Buffers and Files" describes the fundamental procedures and commands for saving the contents of a buffers to a file and later retrieving those contents.

The descriptions are segmented and brief so you can quickly find the information you want. Basically, you should skim the descriptions and read specific information as necessary.

The chapters, "Writing and Editing Lisp Code" and "Writing and Editing Other Code", assume fundamental knowledge of text editing. Much of the information about writing and editing text applies to writing and editing source code, and that information is not duplicated later.

Accessing the NMODE Editor

Regardless of how you get there, you edit text in a buffer. Getting into a buffer that has an appropriate editing mode is how you invoke NMODE's EMACS-type editor. Buffers and the Buffers Facility are covered in detail in the chapter "The Buffers Facility". You should be familiar with the information in that chapter, although some of it is duplicated here.

Writing New Text

To write some new text that you plan to save to a file:

Execute the Find File command, **C-X C-F**, from anywhere in NMODE. You will be prompted for the name of the file relative to some directory. After you enter the file name and type [Return], a new buffer is created and selected. You are now ready to begin editing and entering text. Note that the file is not created until you explicitly save the contents of the buffer with either the Save File or Write File command.

To create and edit text that you do not initially intend to save:

Execute the Select Buffer command, **C-X B**, from any location in the NMODE environment. On the prompt, enter a buffer name (such as MEMO) and press [Return]. The buffer will be created (if it does not already exist), and selected. If while editing, you decide you want to save the contents of the buffer to a file, execute either the Save File or Write File command. You will be prompted for the name of the file to write the buffer to.

Editing an Existing File

To edit an existing file, either

1. Get into a Directory browser that contains the file. Point to and browse the file. This exits the browser and puts you into a buffer that contains a copy of the file.

2. Execute the Find File command, **C-X C-F**, from any location in NMODE. On the prompt, enter the pathname for the desired file and press Return. As with the above procedure, the command gets you directly into an editing buffer that contains the contents of the file. In this case, if the file does not already exist, the command creates an empty buffer that is associated with the entered file name.

Again, remember that the changes you make to a buffer are not saved to its associated file until you explicitly do so. You can use one of the Write File, **C-X C-W**, or Save File, **C-X C-S** commands; or the `Write` command line command of the Buffers Facility; or the `Utility Save` command line command of a Directory browser.

Establishing an Editing Environment

Writing or editing text typically begins with setting up an environment that is suited to a particular editing session. This can be rather simple, or very complex. In general, you set up an editing environment by coordinating the use of windows, buffers, files, modes, and commands.

There are many ways to coordinate these things, and you do not necessarily perform a fixed set of steps. How you work depends on largely on the type of editing you want to do and your personal style.

The next several subsections describe how to set up an environment for text editing.

Setting Desired Modes for Editing

An editing buffer has Emacs mode and can have additional minor modes. The modes determine the available editing commands and overall functionality. The mode line displays the current situation.

When you get into an editing buffer, the default editing mode is called **Emacs**. This major mode suspends the write-protection usually provided in a browser and provides a character-insert state in which a typed character is entered into the buffer at the current location of the point and is echoed to the screen. The cursor appears on the screen to indicate the next location for character-entry. The mode line for a buffer will display something like this:

```
 I/O @ Emacs (Text Fill) JUNK $HOME/docs/a-text-file.txt  -34%-
```

The mode line tells you that the major mode is `Emacs`. `Text` and `Fill` are minor modes.

Minor Modes

While major modes are mutually exclusive (only one is current at one time) you can also be in some minor modes: Text, Lisp, C, Pascal, Fortran, Auto Fill, HP-UX, and so on. You would not be in all these minor modes at once.

See the chapter on non-Lisp editing for information on C, Pascal, and Fortran minor modes. The `HP-UX` minor mode is described in the chapter called "HP-UX Access".

Within Emacs mode, a minor mode provides an extended or altered set of commands and functionalities that enhance your editing efficiency. Auto Fill mode is usually used with Text mode and is not typically used with other modes.

In general, the commands that alter their functionality to suit the mode relate to:

- Indenting.
- Definitions of what words, sentences, and paragraphs are. These affect commands like Move Forward Word.
- Matching brackets or braces.
- Evaluating source code files.
- Formatting lists, defuns, and forms.

To invoke a minor mode, you execute **M-X** *string* **mode** where *string* is a minor mode such as `Text`, `Lisp`, or `Auto Fill`.

Besides letting you set a minor mode directly, NMODE can set a minor mode. When you browse, visit, or find a file, NMODE uses information in the file's header or the filename suffix (e.g. `txt`, `l`, `c`, `p`, `f`, `sl`) to invoke the mode without intervention on your part. The minor mode is displayed in parentheses in the mode line when you are at the top level. For example,

```
   I/O @ Emacs (Text)
```

indicates a current buffer in Emacs major mode in Text minor mode with no autofilling. Because of this intervention, you will occasionally notice changes in the minor mode from time to time even though you took no overt action to change it.

Auto Fill Minor Mode

Besides invoking minor modes related to languages, you can provide automatic filling of sentences by executing **M-X auto fill mode** when you are in a buffer. When Auto Fill mode is on, NMODE inserts an end-of-line between words at the appropriate point when entry of text approaches the right margin. The term, `Fill`, is displayed in the mode line in parentheses to the right of the major mode when Auto Fill mode is enabled; for example,

```
     @ Emacs (Text Fill)
```

Executing the command again disables the mode. If you have them, certain softkeys and popup menu items let you execute the command.

Assuming that you have established an appropriate environment, the next sections assume you are in a buffer, ready to edit.

Fundamental Text Editing

This section describes the major commands and techniques for editing.

The Cursor and the Point

An editing buffer has a **point**, which is an invisible marker that determines where the next character is inserted. The point moves as you insert characters or do assorted editing. Numerous commands alter the location of the point.

The **cursor** in a pane shows the location of the point. The point is **between** the cursor and the character immediately to the left of the cursor.

Moving the Cursor

A mouse lets you move the cursor to any location within a pane. Position the pointer on a desired character (other than the current character) and click the left button; the cursor moves instantly to the new location. This is handy for moving the cursor to "faraway" locations. In addition, clicking the right button with the pointer inside a pane provides a popup menu related to the current mode. The `Jump >>` command in this menu will let you move the cursor. You will get a new popup window that lets you move forward a word, sentence, or paragraph.

You can use the key sequences in the following table to move the cursor. An **N/A** means the command is not applicable or not available. Most commands in the table are discussed in more detail in other sections.

Type of movement	Move to start	Move to end	Move to previous	Move to next
Character	N/A	N/A	C-B	C-F
Line	C-A	C-E	C-P	C-N
Word	M-B	M-F	M-B	M-F
Sentence	M-A	M-E	M-A	M-E
Paragraph	M-[	M-]	M-[	M-]
Paneful	N/A	M-R	M-V	C-V
Buffer	M-<	M->	N/A	N/A
Lisp form	N/A	N/A	C-M-B	C-M-F
Lisp defun	C-M-A	C-M-E	C-M-A	C-M-E

In relation to the commands in the table, note the following details:

- Keyboards usually have keys that duplicate many of these commands. For example, the cursor movement keys (▲ ▼ etc.) essentially duplicate the **C-F**, **C-B**, **C-N**, and **C-P** commands.

- The down-arrow key does not create a new line when you are at the end of a buffer as does **C-N**. The Move Down Line Extending and Move Up Line commands, **C-N** and **C-P**, use the value of the global variable, `goal-column`, to determine which column to move to. You can set this value by executing the Set Goal Column command, **C-X C-N**. With no argument, **C-X C-N** uses the current column to determine which column will be used by vertical movement commands that move the point up or down regardless of which column the point is in prior to executing the vertical movement commands. To disable goal column tracking, execute **C-X C-N** with a prefixed argument.

- Executing [Prev] and [Next] duplicates the Previous Screen command, **M-V**, and the Next Screen command, **C-V**, respectively.

- Executing [⤓] duplicates the Move To Buffer Start command, **M-<**, and [Shift] [⤓] duplicates the Move To Buffer End command, **M->**.

Should you wish to know where the cursor is in a buffer, execute the What Cursor Position command, **C-X =** , which displays information about the cursor's location in the echo area. The command works differently with a prefixed argument, treating the argument as a line number and jumping to the corresponding line.

Inserting Characters or Text

An editing buffer is in a character-insert state. Just type to insert any valid, single-keystroke, printable character.

- The cursor moves forward one position after inserting any character.

- Typing a character with the point to the left of (before) any existing characters inserts that character and pushes all the following characters forward.

- Press [Return] to end a line. This creates a new line so you can resume typing one line down, beginning at the left margin. Also, pressing [Return] anywhere along a line opens the line, moves any remaining parts of the line down and to the left margin, and leaves the cursor under the character at the beginning of the remaining partial line. In contrast, executing the Open Line command, **C-O**, opens the line, moves the remaining line down and to the left margin, and leaves the cursor at the point of opening on the original line.

- The system displays a `!` as the rightmost character of a line when the line is too long to fit in a pane. You can use any of the cursor motion commands to move the cursor to the text that is not visible. The window will scroll horizontally as necessary. The Scroll Window Right, **C-X >**, and Scroll Window Left, **C-X <**, commands can be used to change a pane's perspective on a buffer.

- All printing characters insert directly. Other characters act as editing commands and do not insert themselves. To insert a Control, Escape, Backspace, or Rubout character, escape the character by prefixing it with the Insert Next Character command, **C-Q**. The inserted character is preceded by a ^ on the display; for example, ^D to differentiate the character from a normal printing character.

Inserting Blank Lines

Insert blank lines by executing **C-O** when you want to insert a line between lines and then type the line. Prefix the **C-O** with an argument to create a specified number of blank lines. After you insert or edit some text, all but one extra line can be deleted by executing the Delete Blank Lines command, **C-X C-O**.

Deleting Characters or Text

You can delete characters and small blocks of text by executing any of the following commands.

- Pressing Back space deletes the character to the right of the cursor. If the character is a tab, it is expanded into spaces, and then one of the spaces is deleted.
- The Delete Forward Character command, **C-D**, deletes the character under the cursor, which is just left of the point.
- The Delete Blank Lines command, **C-X C-O**, deletes spaces, tabs, and blank lines. If the cursor is on a blank line, all surrounding blank lines will be deleted, but the line with the cursor will remain. If the cursor is on a non-blank line, all continguous blank lines below it will be deleted.
- The Delete Horizontal Space command, **M-\**, deletes contiguous horizontal spaces on a line on either side of the point.
- The Delete Indentation command, **M-^**, deletes the line feeds that created blank lines and the indentation at the beginning of the current line, leaving one space.

In addition to these commands, a later section describes commands for deleting specific text items; for example, a word, line, sentence, marked region, or buffer.

Items You Can Edit

The NMODE editor recognizes the following items as logical units. Commands are provided to manipulate text in these units.

Character	Printing characters such as A B Z 0 1 2 9 * ^ $ \ # @ (| {) ! ? , ; : ' ".
Word	What a word is depends on the minor editing mode. Basically, there are characters that are considered to be part of a word, and there are characters that are considered to come between words.
Line	The set of horizontally aligned characters in a buffer is a line.
Sentence	Sentence is defined in the subsequent section called "Sentence Commands".
Paragraph	Paragraph is defined in the subsequent section called "Paragraphs Commands."
Region	A region is all the characters between a mark and the point. The "length" of a region can vary from one character to an entire buffer. The point can be previous to the mark, or vice versa.
Buffer	There are commands that apply to the entire buffer.

These items can be manipulated in various ways by using the commands and procedures discussed in the next several sections.

Word Commands

Several commands let you manipulate words. What's a word? It depends on the current mode, but in general a word is any contiguous string of **constituent** characters delimited on both ends by a non-constituent character. In Emacs mode, constituent characters are letters, numbers, and the characters - _ '. In Lisp minor mode, the following additional characters are constituent: `! $ % & * + - . / < = > ? @ [ ] ^ { } ~ # \`. An example shows the implications. You can execute **M-D** to delete a word. Executing the command with Lisp mode off and the cursor under the b in `big-sad*wolf` only deletes the `big-sad`, leaving the ***wolf**. In Lisp mode, however, the whole string is deleted.

Here are the commands for editing words with some supplemental explanation. By convention, commands for words are Meta commands.

M-D	The Delete Forward Word command deletes characters from the cursor location to the end of the current word.
M-Backspace	The Delete Backward Word command deletes characters from the character to the left of the cursor back to the beginning of a word.
M-F	The Move Forward Word command moves the cursor forward over a word.
M-B	The Move Backward Word command moves the cursor backward over a word.
M-T	The Transpose Words command switches the word beginning at the point with the word to its left.

Note that the commands for words are the same as the commands for characters, except that Meta is used instead of Control.

Each command deletes from the point to the beginning or end of a word. For example, if the cursor is under **s** in **makeshift** and you execute **M-D**, the **shift** and any following white-space is deleted, leaving the cursor just after **make**. Since the words are deleted, they can be yanked, which is discussed later in a section called "Yanking (Reinserting)".

Punctuation marks between the point and the beginning or end of a word are not bypassed. For example, if the cursor is under the **t** in **time?zone** and you execute **M-F**, the cursor moves forward to rest on the **?**; it does not move past the **e** in **zone**.

Line Commands

Several commands let you manipulate lines. A **line** is a set of horizontally aligned characters.

Here are commands for editing lines:

M-M	The Back To Indentation command moves the cursor back to the indentation for the current line.
M-S	The Center Line command centers the current line.
C-X C-O	The Delete Blank Lines command deletes blank lines according to three cases: (1) Spaces, Tabs, and succeeding blank lines when the cursor is on a nonblank line; (2) Previous and succeeding blank lines when the cursor is on a blank line; and (3) Blank lines beyond the cursor when cursor is at the end of a nonblank line.
M-X Delete Matching Lines	Deletes all lines after the point that contain a particular string. You are prompted for the string.
M-X Delete Non-Matching Lines	Deletes all lines after the point that do not contain a particular string. You are prompted for the string.
C-K	The Kill Line command deletes the current line from the location of the cursor to the end of the line and leaves the line open.
C-N	The Move Down Extending command moves down to the next line, extending. The term **extending** means that, if necessary, the command will insert a newline in the buffer so the cursor can move down to the next line.
▼	The Move Down command moves down one line, except at the end of a buffer.

C-P	The Move Up command moves up one line. You can also press $\boxed{\uparrow}$.
C-E	The Move To End Of Line command moves to the end of the current line.
C-A	The Move To Start Of Line command moves to the start of the current line.
M-X Count Occurrences	Prompts for entry of a string and then displays in the message area the number of lines that contain the string.
C-O	The Open Line command does several things. First, it "opens" or clears out the current line from the cursor's location. It also inserts any existing text after the cursor on the next line, beginning at the left margin. Finally, the command pushes after lines forward and leaves the cursor after the opening on the current line, ready for you to type additional text.
$\boxed{\text{Insert line}}$	The Open Line Indent command opens a line under the current line and indents so that the cursor is in the same column in the newly opened line as it occupied on the current line. Existing forward lines move down.
C-M-O	The Split Line command splits a line at the cursor's location in the current line, moves the remainder of the line down one line and indents it to the column that was occupied by the cursor. The cursor is placed at the beginning of the remainder of the line.
C-X C-T	The Transpose Lines command transposes the current and above lines and moves the cursor down to the next line, except when the current line is the last line of a buffer.

Sentence Commands

The following commands let you work with sentences. A **sentence** is a string of characters that begins at:

- The start of a buffer, or
- The start of a paragraph, or
- First non-blank after a preceding sentence.

A sentence ends at:

- The end of a buffer, or
- The end of a paragraph, or
- A sentence terminator (. ? !) followed by zero or more sentence extenders (' ")]), followed by at least two spaces.

A sentence can have any number of parentheses, braces, brackets, apostrophes, or quotation marks and other special characters in between. Since this is a rather "long" definition, here is an example that assumes spaces on both ends:

```
This is an {example} of a rather strange [sentence] with a "Lisp
'form'", (+ (* 2 3)(+ 4 5)), in it.
```

M-A	The Backward Sentence command moves back to the beginning of a sentence.
M-E	The Forward Sentence command moves forward to the end of a sentence.
M-K	The Delete Sentence command deletes forward from the cursor's location to the end of the sentence, including special characters and ending punctuation.

C-X Backspace	The Backward Delete Sentence command deletes backward from the cursor's location to the beginning of the sentence. This includes special characters but does not include the spaces before the sentence.
C-M-T	The Transpose Sentences command transposes the current sentence with the previous one.

Notice that the commands for sentences parallel the commands for lines. For example, **M-A** for sentences parallels **C-A** for lines.

Paragraph Commands

The following commands let you work with paragraphs.

In Text mode, a paragraph is delimited above and below by a blank line, but this is not the complete picture. Many formatting packages recognize a line or sentence that begins with a period as a formatting command. Such lines or sentences are treated as a text formatting command line and are not treated as part of any paragraph. While this is true, it can be difficult for the command to account for all situations. It is usually better to use blank lines to delimit paragraphs and insert formatting commands so that they do not get "lost" when you edit a paragraph.

In a language mode, a **paragraph** is a set of characters that is delimited above and below by a blank line so that the commands for paragraphs work, even though, in the reality of programming, there is no paragraph per se.

M-[	The Backward Paragraph command moves the cursor back to the beginning of the current paragraph.
M-Q	The Fill Paragraph command fills the paragraph that contains the cursor.
M-]	The Forward Paragraph command moves the cursor forward to the end of the current paragraph.

C-M-P	In Text mode, the Set Paragraph Indent command lets you set the value for paragraph indentation. Notice that, in Lisp mode, this key sequence is bound to the Move Backward List command, which moves the cursor backward to the beginning of a list.
M-X mark paragraph	The Mark Paragraph command lets you make the current paragraph into a region by placing a mark and the point around the current paragraph. The cursor moves to the beginning of the paragraph.
C-X M-T	The Transpose Paragraphs command transposes the current paragraph with the previous one.

Prefixed Arguments (Repeat Factors)

Many of the commands used to edit text accept prefixed arguments supplied by the Universal Argument command, **C-U**, or **C-U** *integer*. This command was discussed in Chapter 2 and is also described in the *NMODE Command Reference* manual. In most cases, you just prefix a command to cause the command to execute a specified number of times. For example, **C-U 34 M-D** deletes 34 words, beginning at the location of the cursor. As another example, executing **C-U 10 C-O** opens ten lines below the current location of the cursor.

A negative command argument will usually cause a command to execute "backwards". For instance **C-U -1 M-D** will delete the word to the left of the cursor instead of to the right.

Undoing Text

If you are in an editing buffer other than the OUTPUT buffer or an HP-UX shell buffer, you can undo changes by executing the Undo command, **C-X U**. When you modify a buffer's contents, NMODE uses a stack for each buffer to store text that you insert or delete with editing commands such as Delete Word and Delete Line. Thus, when you edit something and then decide you want to undo it, executing **C-X U** undoes whatever you did.

If the stack has more than one entry, the message area displays:

```
Hit space to undo more:
```

Repeated pressing of the space bar will undo the available sequence of edits on the stack, where each stack entry will undo one NMODE command. Pressing any other key exits the undo mode and does whatever the key does. To just exit the Undo command, press ESC.

When the stack is empty, and there is no more history of entries, the message area displays:

```
Nothing Left to Undo
```

If there is no undo stack for the current buffer, the message areas displays:

```
Sorry, No undo history on this buffer.
```

The Undo command can even undo the effects of a previous Undo command. For example, If you are undoing a sequence of commands, and realize that you really did not want to undo, simply leave the Undo command with ESC, and then execute the Undo command, **C-X U**, again.

In general, all editing commands can be undone, except for commands that overwrite the entire buffer with new data from a file, like the Visit File and Revert File commands.

You can execute **M-X set undo depth** to alter the default undo-history stack depth for the current buffer; the default is 50 commands. The command prompts for entry of a new undo depth. Note that the Set Undo Depth command destroys the old undo history stack, so changes done before cannot be undone. Setting the depth to 0 disables the undo feature for the current buffer. In the NMODE General User Options there is an entry for "Undo Stack Depth For New Buffers". This controls the undo depth for any new buffers you create, but does not change any existing buffers.

Miscellaneous Commands

The Cursor Location

To get precise information about the current location of the cursor, and consequently about the location of point, execute the What Cursor Position command, **C-X =**. The command displays something like the following information in the echo area:

```
X=3 Y=19 CH=40 line=428 (74 percent of 574 lines)
```

The displayed fields have the following meanings.

- The `X` value is the column (zero is the leftmost column).
- The `Y` value is line number in the pane that the cursor is on (zero is at the top).
- The `CH` value is the ASCII value of the character above the cursor (after the point).
- The `line` value is the line number that the cursor is on in the buffer.
- The percentage shows the relationship between the location of `line` and the total number of lines in the buffer; for example the line can be `74 percent` of the way through the buffer.

Case Conversion Commands

Several commands let you convert some type of character, word, or range of characters into upper or lower case. Here are the commands.

M-L	The Lower Case Word command converts the following word to lower case and moves the cursor past the word.
M-U	The Upper Case Word command converts the following word to upper case and moves the cursor past the word.
M-C	The Upper Case Initial command capitalizes the following word and moves the cursor past the word.
C-X C-L	The Lowercase Region command converts the current region to lower case, but does not move the cursor or mark.
C-X C-U	The Uppercase Region command converts the current region to upper case, but does not move the cursor or mark.

M-' The Upcase Digit command converts the last digit typed to its shifted character; for example, the command converts **5** to **%**. Notice that the first time you execute the command during an editing session, the command prompts you to type the digits `1, 2, ..., 9, 0` while you hold down Shift and then converts the digit. This "teaches" the command the characters that correspond with the digits on your keyboard.

The word case commands accept arguments. With a negative argument, the commands work backwards on words as specified by the argument, but do not move the cursor. Actually, the commands work from the location of the cursor in a word. For example, executing **M-L** with the cursor under the **F** in **YOURFILE** changes the word to **YOURfile.** Prefixing the commands **M-L**, **M-U**, and **M-C** with an argument of **-1** causes them to act on the previous word, leaving the cursor where it was.

Transposing Text

While writing or editing, you often want to alter the sequence of certain items. The following commands let you transpose text.

- The Transpose Characters command, **C-T**, transposes the characters either side of the point. Visually, this is the character above the cursor and the character to its left. Executed at the end of a line, the command transposes the two characters to the left, not the ending line separator and the character to its left. This command is usually used to fix simple typos.
- The Transpose Words command, **M-T**, moves the cursor forward over a word, dragging the "appropriate" word forward as well. If the cursor is just before the word, the command drags the preceding word forward. If the cursor is on a word, that word is dragged forward past its forward word. There are three possibilities for using a prefixed argument.
  - An argument of zero, transposes the word at the point (surrounding or adjacent to it) and the word at the mark.
  - A positive argument serves as a repeat count, dragging the appropriate word forward over the specified number of words. When the cursor is on a word, that word is dragged forward. If the cursor is between words, the previous word is dragged forward.
  - A negative argument serves as a repeat count, dragging the appropriate word backward over the specified number of words. When the cursor is on a word, that word is dragged backward. If the cursor is between words, the previous word is dragged backward.

In any case, a punctuation or delimiter character between two words does not move. For example, **FOO, BAR** transposes to **BAR, FOO** and **human-like (humanoid)** transposes to **humanoid (human-like)**.

- The Transpose Lines command, **C-X C-T**, transposes the current line and the previous line, and moves the cursor down one line.
- The Transpose Regions command, **C-X T**, is a general purpose command that lets you transpose two regions, which do not need to be adjacent. Here is one way to do it:

  a. Move the cursor to the beginning of the first region and execute the Set Mark command, **M-Spacebar**, to set a first mark at the beginning of the region. Then, move the cursor to just past the end of the first region and execute the command again to set a second mark.

  b. Move the cursor to the beginning of a second region and set a third mark. Then, move the cursor to just past the end of the second region and execute **C-X T** to transpose the regions.

The order in which you set marks is not important and the point (cursor) can be at a beginning or end; you only need have two distinct regions. Use this command and procedure to transpose any two arbitrary blocks of text.

These commands let you transpose blocks within a buffer. A later section discusses block copies and moves within a buffer and among buffers.

Search Commands

These commands help you find words in text.

Incremental Search

The Search Forward command, **C-S**, and the Reverse Search command, **C-R**, are the two major commands for searching for strings. The Reverse Search command, **C-R** works the same way as the Search command, but in a backward direction.

Beginning at the cursor's location, **C-S** reads a typed character and moves forward to the first occurrence of that character. The command continues this forward movement as you type additional characters. The cursor moves during entry of characters, and when you stop typing, the cursor indicates the first occurrence of the accumulating string. This process continues until one of the following states occurs.

- Pressing $\boxed{\text{ESC}}$ terminates a search and leaves the cursor at its current location.
- Appending a character to a string that creates a string which has no forward existence causes the search to fail and indicates in the message area that the I-search is failing.
- Certain keys or commands such as the Backspace key or Insert Next Character command, **C-Q**, apply to the search string in the message area. In particular, pressing the Backspace key backspaces in the search string so you can enter a different character if you like. Executing **C-Q** lets you enter a character which causes the command to search for a non-printing character instead of a printing character.
- Executing the NMODE Abort command, **C-G**, during entry of characters terminates a search and moves the cursor back to where the search started. On a failing search, the command throws away characters that caused the search to fail, backtracks to where the search was successful, and lets you enter different characters. Another execution of the command at this point exits, canceling the entire search.
- Any command not recognized by incremental search will end the search and execute that command.

Repeated Searches

You might need to execute C-S several times to find the "right" occurrence of the string because the command finds the first occurrence. This might not be the string you want. To search again, execute C-S to start the search and then, with no entry of characters, execute C-S again to set a state that means: search again for the string used previously. You can repeat this successively until you find the "right" string.

Case-Sensitive Searches

The searches ignore case. To enable a case-sensitive search, execute C-C after executing a C-S or C-R and before you enter a character. So doing displays the message: `Case sense is ON:`

Word Search

The Word Search command, C-M-S, prompts for entry of the word you want to find. If you previously executed the command, the command offers the previous word as a default. Accept the default or enter a word and press [Return]. Then, the command either: finds the word and moves the cursor just after it; or does not take any apparent action and leaves the cursor at its current location. In particular, no message appears in the message area to indicate that the word does not exist.

Multi-line Patterns

Incremental search will match multi-line patterns. That is, XYZ [Return] will search for XYZ at the end of a line. Searching for [Return] XYZ will find occurrences of XYZ at the beginning of a line.

String Replacement Commands

The Replace String command, **M-X replace string**, prompts for the name of a string to be replaced, prompts for the name of a string to be inserted, and performs the replacement operation forward from the cursor's location to the end of the buffer, indicating the number of replacements. The cursor will not move

Query Replace Command

The Query Replace command, **M-X query replace** or **M-%**, works about the same way as the Replace String command, except that on each potential replacement, the command provides the following options:

Y	Perform the replacement operation.
N	Skip replacement and move to the next possibility.
,	Perform the replacement operation and display the result, which lets you inspect the replacement and then enter a **Y**, ESC, !, or ^ character.
ESC	Exits without finishing the replacement operation.
.	Perform the current replacement operation and then exit.
!	Perform all replacements without intervention.
^	Go back to the previous replacement possibility, whether the replacement was made or not.

Entering any other character or executing another command terminates the replacement operation and inserts the character or executes the command.

Executing **M-X count occurrences** prompts for a string and then counts the number of occurrences of the specified string in the buffer from the cursor's location to the end of the buffer. The count is displayed in the message area.

Executing **M-X delete non-matching lines** prompts for entry of a string and then deletes all lines from the cursor's location to the end of the buffer that **do not** contain the string.

Executing **M-X delete matching lines** prompts for entry of a string and then deletes all lines from the cursor's location to the end of the buffer that **do** contain the string.

Formatting Commands

These commands are useful when formatting text.

Setting Margins and Indenting

You can further configure an environment for editing by setting assorted parameters related to margins.

- In Text mode:
  - The Set Left Fill Column command, **C-M-Y**, lets you specify a left margin. The default setting is 0.
  - The Set Paragraph Indent command, **C-M-P**, lets you specify the indentation for beginning a paragraph. The default setting is 5.
- In any mode:
  - The Set Fill Column command, **C-X F**, sets the right margin. You can prefix the command with the Universal Argument command, **C-U** *integer*, to set an arbitrary right margin; for example, **C-U 72 C-X F** sets the right margin at 72. Alternatively, you can move the cursor to the desired column and execute **C-X F**.
  - The Set Goal Column command, **C-X C-N**, lets you set, or flush, a global value for horizontal positioning of the cursor as you move from line to line. In any line, position the cursor on the column you want the cursor to move to when you move to a line and execute **C-X C-N**. To unset the goal column, execute the command with a prefixed universal argument.
  - The Set Fill Prefix command, **C-X .**, makes the characters in the current line from the left column to the current column into a fill prefix by assigning the string as the value of the system variable `fill-prefix`.

Tabbing and Indenting

The exact way in which tabbing and indenting occurs varies, depending on which mode you are in, but the principles are consistent. The language modes: Lisp, C, Pascal, and Fortran effect tabbing and indenting according to conventions for writing source code in the particular language. Text mode provides straightforward tabbing and indenting.

The next few subsections discuss schemes for tabbing and indenting. Briefly, here is a list of the available commands.

Tab	Indents "appropriately" according to current mode.
M-I	Indent to tap stop.
M-Tab	Inserts a Tab character, which usually tabs 8 spaces.
C-M-	Indents an entire region to the same arbitrary column; for example, ten spaces. (e.g. **C-U 10 C-M-\**
C-X .	Sets a fill prefix consisting of the string from the left column to the current column; for example, >>>.
M-^	Appends current line to the line above, undoing effects of other indenting and tabbing.
M-	Deletes space and tab characters about the point horizontally.
M-M	Moves the cursor back to the current line's first nonblank character, which means that it moves it back to the indentation.

Tabbing

Tabbing varies according to established conventions for the current mode. In text mode, the cursor moves forward 8 spaces. In a language mode, the cursor moves according to conventions for the current language.

The actual key you press to do tabbing can vary depending on which hardware you have and which mode you are in. Here are some guidelines:

- In Text mode, you press Tab if your system has default key bindings and left Extend char is the Meta key.
- If you customized your system or your system does **not** use Extend char as the Meta key, then you might **not** press Tab to tab. You can always tab by executing the Tab To Tab Stop command, **M-I**, which inserts a Tab character.

- In any language mode, executing **M-I** indents appropriately, according to the mode: Lisp, C, Pascal, or Fortran.
- In any mode, text or language, executing **M-Tab** or **C-Q Tab** also tabs by inserting a Tab character, which is 8 spaces.

Indenting

In English text, you often indent the first line in a paragraph and do not indent subsequent lines. When you are in Text and Auto Fill modes, the Set Paragraph Indent command, **C-M-P**, lets you establish a column value for indenting the first line. To set the value, either: execute the command with the cursor on the desired column for indentation; or prefix the command with the desired column value. For example, **C-U 5 C-M-P** sets paragraph indentation at 5 spaces. To remove the 5-space indentation, execute **C-U 0 C-M-P**, or move the cursor to the left margin and execute **C-M-P**.

The established value for indentation is used by the Fill Paragraph command, **M-Q**. The value is also used when you begin a new paragraph when Auto Fill mode is on.

Centering

The Center Line command, **M-S**, centers a line according to the length of the line in relation to the value assigned to the system variable, `nmode:fill-column`. Prefixing **M-S** with a negative argument centers a specified number of lines above the current line, leaving the cursor on the current line. Prefixing the command with a positive argument centers the specified number of following lines, beginning with the current line, leaving the cursor beyond the centered lines.

Filling

Enabling Auto Fill mode by executing **M-X auto fill mode** lets you type text or source code that is **filled** as you type. That is, the text or source code is broken up into lines that fit in a width specified by the left and right margins. This works fine when you type and do not immediately edit the text.

Later editing can create text that is unequal, scrolled, or contains lines that appear to end with a ! character. The ! means that the line extends past the visible display. The next few subsections explain how to fill text.

General Commands for Filling

Here are the major commands or keys for filling.

Spacebar	When Auto Fill mode is on, breaks lines at the right margin when the location of a word will exceed the value for `fill-column`. This action continues to the end of the paragraph.
$\boxed{\text{Return}}$	Has the same effect as space before it creates a new line.
M-Q	The Fill Paragraph command fills the current paragraph, using values for `fill-column`, `left-fill-column`, and `paragraph-indent`. Be sure that blank lines delimit the paragraph! Otherwise, you can scramble your text, especially if it contains characters and commands for formatting.
C-U M-Q	Fills and justifies a paragraph. Note that this is **M-Q**, which fills a paragraph, prefixed by the Universal Argument command with no argument.
M-G	The Fill Region command fills a region.
C-X =	The What Cursor Position command displays information about the current cursor position in the message area.

System Variables Related to Filling

Some filling occurs in accordance with the values given to the system variables: `nmode:fill-column`, `nmode:left-fill-column`, and `nmode:paragraph-indent`. The latter two variables apply to filling done in Text mode. The defaults, respectively, are 70, 0, and 5. You can set the values by executing the following commands, respectively:

C-X F	The Set Fill Column command sets the fill prefix at the column that contains the point, or it sets the fill prefix to a value specified by a prefixed argument. For example, **C-U 70 C-X F** sets the value of `fill-prefix` to 70.
C-M-Y	The Set Left Fill Column command sets the left fill prefix. Use the same procedure as for fill prefix.
C-M-P	The Set Paragraph Indent command sets the value for paragraph indentation. Use the same procedure as for fill prefix. Only the first line in a paragraph is indented.

Filling Paragraphs

After you edit a paragraph and wish to fill it, position the cursor anywhere in the paragraph and execute the Fill Paragraph command, **M-Q**. Execute **C-U M-Q** to fill and justify the paragraph. Putting the cursor anywhere in the paragraph is equivalent to putting the point inside the paragraph. If the cursor (point) is between paragraphs, the command fills the next paragraph.

Filling Regions

The Fill Region command, **M-G**, fills each paragraph in the region.

Beware of Filling with Formatting Commands

The **M-Q** and **M-G** commands fill an entire paragraph or region. This is not a problem when you just type text, but if you insert special commands that are subsequently used to format the text during printing, be sure to separate the commands for formatting from the text by temporarily inserting a blank line above and below the text you intend to fill. Otherwise the formatting commands will be treated as normal text and filled.

Using a Fill Prefix

When you are in Auto Fill mode, in some cases, you might want to begin each line in a paragraph with a specific string; perhaps something like the string:

```
Ace Company Policy:
```

You can use the Set Fill Prefix command, **C-X .**, to assign a string to the system variable, `nmode::fill-prefix`. Then, the string is inserted at the beginning of each new line. Here is an example that illustrates the process:

1. With the cursor at the begining of a blank line, type:

   ```
   Ace Company Policy:
   ```

   and execute **C-X .**

2. Continue typing; for example, type something like:

   ```
   Coffee is available at no charge.
   ```

   and press [Return]. Notice that the fill prefix is inserted automatically at the beginning of the next line and the cursor is positioned just after the prefix so you can type the next policy.

Here are some fine points related to using fill prefixes.

- You can think of the fill prefix as a marker of sorts. It can be any valid string. Filling a paragraph removes the marker from each line, fills the paragraph, and puts the marker back in each line.

- On continuous typing when Auto Fill mode is enabled, the fill prefix is inserted at the beginning of each new line. It is important to realize this. In relation to the example, if you typed a policy statement that exceeded the right margin, the ensuing filling caused by Auto Fill mode would insert the prefix in the policy statement, which is probably not what you want.

- Any line that does not start with the fill prefix delimits a paragraph.

- To remove a fill prefix, execute **C-A**, or equivalent command, to position the cursor at the beginning of a line. Then, execute **C-X** .. This removes the prefix by setting the value of `fill-prefix` to a null string.

Text Manipulation

Previous sections discussed fundamental editing tasks: moving the cursor, inserting text, manipulating paragraphs, finding strings, and so on. This section describes procedures and commands for transposing two regions or inserting text from disassociated buffers into the current buffer.

Setting and Using Marks

Earlier sections mentioned that the point and something called a mark could establish a region that could represent certain items that could be edited. This section describes how the **point** and a **mark** can be used to establish a region. Many of the commands make use of the regions delimited by the mark and the point.

The Ring of Marks

NMODE can "remember" 16 locations at which you or the system set a mark. Commands that set a mark push any new mark onto this stack and remove the oldest mark when the current stack has 16 marks.

To set a mark, execute **C-Space** (or **C-@**, or **M-Space**). Alternately, you can execute **M-X set mark**.

To return the cursor to the most recent mark, execute the Set Mark command with a "null" universal argument (e.g **C-U C-Space**). This moves the point to where the mark was and restores the mark from the stack of former marks. Doing this repeatedly moves the point successively to all the old marks on the stack and can eventually put the point back where you started. Insertion and deletion can cause the marks to "drift", usually not far unless you insert or delete large amounts of text above the marks.

Establishing a Region

In general, any command that processes an arbitrary part of a buffer must know where to start and stop. Such commands operate on a region, which is a set of characters between the point and the current mark.

Recalling that the point sits just to the left of the cursor, the following procedure is a reliable way to establish a region:

1. Move the cursor under the character at the location where you want to start a region and execute the Set Mark command, **C-Space**, to set a mark. This sets a mark at point, which is just before the first character in the region.

2. Move the cursor forward just past the character where you want to end a region. This leaves the point just before that character, which is after the last character in the region.

You now have an established region. By knowing clearly what is in and not in your region, you can execute commands related to regions with confidence. For example, you can execute the Copy Region command, **M-W**, which makes a copy of a region, and then yank the region into another part of the current buffer with complete confidence that the region will be inserted as desired.

It is almost as easy to set a mark and move the cursor backwards as long as you keep the point-cursor relationship in mind so you know exactly what is and is not in the region.

An Example of Using Marks and Regions

Here is an illustrative example. Assume the following text:

```
The ACE Company's new policies appear to
be effective.  Anticipating an increase in
the use of AI techniques, Ms. Trundle assigned
ace expert and dedicated hacker, Julie Smart, the
task of teaching the programmers, who understood
Pascal, how to write Lisp code.
```

Then, assume you set the following marks:

1. Position the cursor under the `A` in `Anticipating` and set a mark, which places a mark just to the left of the `A`.
2. Now, move the cursor to the top of the text and execute the Delete Region command, **C-W**. Since the point sits just to the left of the cursor and the first available mark sits just to the left of the `A`, the following text disappears.

   ```
   The ACE Company's new policies appear to
   be effective.
   ```

3. Now, position the cursor under the `J` in `Julie` and execute **C-Space** to set a mark.
4. Move the cursor under the comma just after `Smart` and execute **C-W** to delete the current region. Since the point sits just to the left of the comma and the current mark is just to the left of the `J` in `Julie`, the following text is deleted.

   ```
   Julie Smart
   ```

There is no point to this example other than to help you visualize how establishing certain regions lets you control text manipulation of items other than characters, words, and sentences. NMODE has several commands for working with regions. They are listed in Appendix A and are discussed further in this manual in a later section called "Deleting, Copying, and Moving Blocks of Text"

Commands for Setting Marks

Here is a list of commands that let you set marks.

C-Space	The Set Mark command sets a mark at the current location of the point.
M-@	The Mark Word command sets a mark after the end of the next word. The point does not move. In effect the command, makes the next word a current region. Place the cursor between words because, when the cursor is on a word, the mark is set at the end of the current word, possibly not what you want.
C-<	The Mark Beginning command sets a mark at the beginning of the current buffer. The point does not move.
C->	The Mark End command sets a mark at the end of the current buffer. The point does not move.
C-X C-H	The Mark Whole Buffer command establishes the entire buffer as a region, setting the mark at the end of the buffer and placing the point at the beginning.
M-P	The Mark Paragraph command establishes the current paragraph as a region, setting the mark at the end of the paragraph and placing the point at its beginning.
C-M-H	The Mark Defun command establishes the current Lisp defun as a region, setting the mark at the end of the defun and placing the point at its beginning.
C-M-@	The Mark Form command sets a mark after the end of the next Lisp form. The point does not move. Again, the cursor should be between forms as is the case for words.

Many NMODE commands affect the point and any marks that might be set. Do not assume too much in attempting to use them. In general, the best procedure is to set marks and use them immediately, before you forget where they are or before an intervening command changes them.

The Exchange Point And Mark command exchanges the current mark and point. This command is useful for verifying the current region visually. Executing it a few times moves the cursor to both ends of the region so you can see exactly what constitutes the region you want to manipulate.

Deleting, Copying, and Moving Blocks of Text

NMODE maintains a "buffer" for the temporary storage of "killed" or deleted text. The buffer is called the **kill ring** in which up to 16 blocks of killed text are saved. Switching buffers has no effect on the kill ring. Even when you visit or find files, the kill ring is unchanged. Consequently, you can do cut and paste operations in a buffer or among buffers and files.

In most cases, each delete operation pushes a new block of text onto the kill ring. But executing several similar kills sequentially combines the text into a single entry on the ring. For example, executing **M-D** 10 times to kill 10 consecutive words puts the ten words onto the kill ring as one entry. Such a block of text can be reinserted by one execution of **C-Y**. Commands that kill forward from the cursor add text onto the end of the text in kill ring, and commands that kill backward from the cursor add text onto the beginning of the text in the kill ring. This way, any sequence of mixed forward and backward kills puts all the killed text into one entry without rearrangement. This sequential execution of a kill command works quite well for words, lines, and sentences in Text mode and for lists, defuns, and forms in Lisp mode.

A kill command that is separated from the last kill command by some other command creates a new entry on the kill ring, unless you execute the Append Next Kill command, **C-M-W**, before you execute the kill command. Executing **C-M-W** establishes a state that tells the following command, if it is a kill command, to append the killed text to the last killed text instead of making it a new entry on the kill ring. The Append Next Kill command lets you kill several separated segments of text, accumulating them until you want to yank the single item into some location.

Some commands for killing or moving text relate to items such as words, lines, sentences, and forms (in Lisp mode). Any other segment of a buffer from a single character to the whole buffer can be established as a region and then killed.

Text does not have to be killed to be put in the kill ring, it can be copied there by executing the Copy Region command, **M-W**, and then yanked. This command actually places the region on the kill ring. That is, it copies the region onto the kill ring so it can be yanked, but it does not kill the existing text.

The Yank Last Kill command reinserts the text of the most recent kill. You can prefix **C-Y** with an argument that will reinsert killed text according to the argument; for example, **C-U 3 C-Y** reinserts the third most recent kill. This is very useful when you can remember the content of the last several kills. Executing **C-Y** puts the cursor at the end of the insertion and puts a mark at its beginning.

Regardless of what text you kill or copy or how you kill or copy it, text is yanked by executing the Yank Last Kill command, **C-Y**, or the Unkill Previous command, **M-Y**. This latter command replaces reinserted killed text with the previously killed text, rotating the kill ring backwards. Executing **M-Y** enough times can rotate any part of the kill ring to the front so you can get at it, if it is among the last 16 kills.

Deleting Text

Here are commands for deleting text.

M-D	The Kill Forward Word command deletes the current word from the location of the cursor to the end of the word.
M-Backspace	The Kill Backward Word command kills the current word from the character left of the cursor backward to the beginning of the word.
C-K	The Kill Line command deletes the current line from the location of the cursor forward to the end of the line, including punctuation and special characters. See the command reference for details.
M-K	The Kill Sentence command deletes forward from the location of the cursor to the end of a sentence. See the command reference for details.
C-M-K	In Lisp mode, the Kill Forward Form command deletes the current Lisp form from the location of the cursor to the end of the form, not including the close paren.
C-M-DEL	In Lisp mode, the Kill Backward Form command deletes the current Lisp form from the character left of the cursor's location to the beginning of the form, not including the open paren.
C-W	The Kill Region command deletes the current region from the point to the current mark. Note that the exact character deleted near the cursor can vary depending on whether you delete forward or backward.

Most of these commands accept prefixed arguments. Their use is discussed in the *NMODE Command Reference* manual if you need additional information.

Using Registers for Insertion

Another option for storing text temporarily and inserting it as desired is to establish a region and execute the Put Region command, **C-X X**. This temporarily "puts" the region into a register. Later, you can insert the region into a buffer at the cursor's location by executing the Get Register command, **C-X G**. A register is associated with a single alphabetic character or digit, which you enter, so you can store several regions in several corresponding registers. You need only to remember your arbitrary relationship between regions and characters for registers during an editing session.

Accumulating Text in Buffers or Files

There are several ways to accumulate and manipulate assorted text besides using the Insert Kill Buffer or Unkill Previous commands.

Any region can be inserted into a buffer by: executing **M-W** to copy the region, invoking a different buffer, and then executing **C-Y** to yank the region to the new buffer. Alternately, you can execute the Append To Buffer command, **C-X A**, which inserts a copy of the region into a specified buffer, at the buffer's end. The command even creates a new buffer if necessary. The cursor moves to the end of the insertion; so repeated appends let you accumulate text. Finally, the Write Region command, **M-X write region**, lets you write the contents of the current region in a buffer to a file.

Any text accumulated in a buffer can be inserted into any other buffer by executing **M-X insert buffer**. In the current buffer, executing the command prompts you to enter the name of the buffer in which you accumulated text and then inserts the entire buffer, beginning at the current location of the cursor.

Any text in a buffer can be appended to a file by executing **M-X append to file**, which prompts for a file name and then appends the contents of the current region of the buffer to the end of the file. Executing **M-X prepend to file** works the same way, except that the contents are added to the beginning of a file.

Buffer and File I/O

Commands are available for directing the contents of buffers and files from various devices or resources to various devices or resources. Perhaps these commands should be discussed in a separate chapter, but the fact is that you typically use them often during the text editing process and as you manipulate code. Consequently, they are discussed here, and you should realize that they apply equally to the later chapters about working with Lisp and other code.

Remember that you can abort most of the following commands by executing **C-G** during entry of strings or file names related to prompts. The commands discussed in the next several subsections appear in an approximate order in which you might use them during an editing session.

Printing a Buffer

You might often want to get hardcopy of text you are editing for study or reference. The Print Buffer command, **C-X C-P**, lets you dump the contents of the current buffer to a specified printer. The specified printer defaults to the HP-UX command *lp*, but you may change the default in NMODE General User Options. The way you specify a printer is to give the name of an HP-UX command that accepts input from standard in. The Print Buffer command simply pipes the contents of the current buffer into the specified command. Note that the HP-UX command is not required to have anything to do with printing hardcopy. For instance,

```
    sort | expand > /tmp/sorexfile
```

Writing a Region

When you are writing or editing text and want to save some part of the text, the Write Region command, **M-X write region**, lets you write the contents of the current buffer's established region to a file, which you specify in response to a prompt.

Finding or Visiting a File

Finding a file or visiting a file are primary initial tasks in the text editing process. Executing the Find File command, **C-X C-F**, lets you read a file into a newly created buffer, which is given the file's name, less its pathname. If the specified file is already in a buffer, the command reselects that buffer.

The Visit File command, **C-X C-V**, is somewhat different because it visits a file in the current buffer, reading the specified file into the buffer and making it the associated file. Be aware that the command replaces the current buffer's contents with the contents of the visited file. Unless you save the current contents, you could lose them. After prompting for the name of the file to be visited, you are asked if you wish to save the current contents of the buffer. If you enter YES, the command prompts for a file name, saves the contents to the specified file, and then visits the new file. In contrast, The Find File command always finds or creates a different buffer, leaving the contents of the current buffer undisturbed. Note that the action of the Visit File command cannot be reversed with the Undo command.

Executing either command prompts you for a file name, and might offer a default or suggest entry of a file name relative to some pathname. When a relative pathname is provided, you need only specify the remaining part of the pathname plus the filename. For example, if the prompt displays:

```
relative to $HOME/tools
```

you might specify something like:

```
mytools/framis.txt
```

so that the entire pathname is:

```
$HOME/tools/mytools/framis.txt
```

By the way, you can execute **M-X set visited file name** to make NMODE think you have changed the name of the associated file without writing to the file, should you need to do so.

Saving a File

Whether you find or visit a file and make changes you want to save, or just enter text into a buffer and decide to save it, the Save File command, **C-X C-S**, saves the contents of a buffer to its associated file. If the current buffer does not have an associated file name, the command prompts you to enter a file name (you can enter an entire pathname), saves the contents of the buffer to the specified file, and makes the file the associated file.

After saving the file, the command indicates that the file is written and displays the number of lines it contains. If no changes have been made in a buffer that has an associated file, the command indicates that there are no changes to save. Saving the contents of a buffer to a file removes the asterisk displayed in the mode line that indicates the presence of changes.

Writing a File

The Write File command, **C-X C-W**, works just like the Save File command, except that you are always prompted for the name of the file to write to.

Saving all Files

The Save All Files command, **M-X save all files**, lets you sequentially save the contents of all buffers that have modified contents. The command finds a buffer that has modified contents and asks you, `Yes` or `No`, to save the file. Entering `Yes` saves the contents of the buffer to its associated file and moves to the next modified buffer. Entering `No` skips the buffer and moves on to the next modified buffer. Execute **C-G** to abort the command and not save any more files.

Creating a File

Besides executing other commands that create files as necessary, you can execute **M-X create file** at any time to create an arbitrary file in any specified directory. On the prompt:

```
Create file whose name is: (Default is: '/users/mike/docs/')
```

where `/users/mike/docs/` is the current relative pathname, enter the name of the file you want to create and press Return. The entry can be a file name relative to the default pathname or it can be an absolute pathname. In either case, the command creates the file and places it in the specified directory, but does not exit your current location and visit the file in a buffer. To see that the file was created, browse into the specified directory and note that it now appears in the list.

Copying a File

The Copy File command, **C-X C-C**, works in a straightforward way to let you make a copy of a file, prompting first for the existing file name and then for the name of the copy. The original file is left intact.

Moving a File

The Move File command, **C-X C-R**, moves a file to another file, changing the actual file name on the disc and deleting the original file. Like copying a file, you enter two filenames, one for the existing file, and one for the moved file.

Killing (removing) a File

The Kill File command, **M-X kill file**, prompts for entry of a file name and then, **irretrievablely, kills the file!** Do not press [Return] unless you are sure you want to remove the file. It is often more efficient to get into an active directory and use the browser's `Trash` command to remove files because the files are not deleted until you exit the browser.

Reverting Back to a File

When you:

1. Visit or find a file;
2. Modify its contents in a buffer;
3. Have not yet saved the changes; and
4. Decide you want to revert back to the original version of the file,

then you should execute the Revert File command, **M-X revert file**.

The command prompts for entry of a file name; you will probably just accept the default, which is the associated file. The command does not change the location of point relative to the beginning of the buffer, but if you made drastic changes in the buffer's (file's) contents before reverting, the location of the point may be nowhere close to its previous location. Note that the effects of the Revert File command cannot be reversed with the Undo command.

Renaming a File

The Rename File command, **M-X rename file**, works just like the Move File command, except that the file must remain in the same directory.

Compiling a File

The Compile File command, **M-X compile file**, lets you compile an existing file of Lisp source code; that is, the file has a `.l` suffix. The command produces a file with an equivalent name, but with a `.b` suffix. You must be in a buffer in Lisp mode. The output from compilation appears in the OUTPUT buffer. On initial execution, the command prompts if you want to save the contents of the current buffer to a file. You should respond YES if you have not yet saved the buffer because the command compiles the file's source code, not the buffer's source code. On being prompted to enter a file, enter a filename with an appropriate suffix of `.l`.

You can be in another language mode and compile an existing source code file with a `.c`, `.p`, or `.f` suffix provided you loaded the program editing support module (`lang-edit`). The compiled file has the suffix `.o`.

Chapter 10
Working with Lisp Code

Introduction

The earlier chapter called "Working With Text" discussed general aspects of text editing. This chapter assumes that information and focuses on how to edit and otherwise manipulate Lisp source code. This includes:

- Automatic indentation and formating of lists, defuns, and forms.
- Automatic matching of parentheses.
- Killing and yanking forms, defuns, and any other arbitrary region.
- Inserting headers, comments, and revisions.
- Convenient methods of interpreting and compiling source code.
- A few very simple means of debugging programs. The *LISP Programmer's Guide* discusses more comprehensive means of debugging source code.

This chapter describes these things in terms of how to use the NMODE commands; it does not deal with how to write programs. Any examples of forms, defuns, or lists are trivial because their only purpose is to illustrate how to use commands.

The information in this chapter is supplemented and extended in the *LISP Programmer's Guide*, which describes programming. In case you need help with Lisp syntax and related information, the *LISP Reference* manual describes the constants, variables, functions, special forms, and macros in Common Lisp.

This chapter discusses the various ways you can do things, but leaves the overall approach up to you. Depending on your familiarity with an EMACS-type editor and the Lisp programming language, you can selectively read sections of this chapter to get the information you need. The sections are:

- "Scrolling and Moving the Cursor" explains how to move around and position the cursor as desired.

- "Writing and Editing Lisp Code" explains finding code, manipulating Lisp code, matching parentheses and such (with a practice session), manipulating headers and comments, and killing and moving code.
- "Executing and Debugging Code" explains evaluating Lisp code, yanking results, simple debugging, compiling source code, and getting more information about programming.
- "Establishing a Programming Environment" explains using packages, loading browsers, creating browsers, creating buffers, and setting margins and prefixes.

To get information, go to a section, skim for what you need, and read as necessary.

Programming usually requires extensive use of facilities which are not re-examined in this chapter. To write, edit, interpret, debug, and compile Lisp source code, you will probably use the Additional Facilities, active directories, code indexes and search indexes.

Like the previous chapter, the sections that follow discuss the commands and procedures in a segmented manner so that you can skim the material and quickly find the information you want.

Scrolling and Moving the Cursor

The general procedures for scrolling and moving the cursor were discussed earlier. This section reviews some information related to editing Lisp code.

When you edit Lisp code, lines can become long and disappear partially on the right side of the screen, displaying an ! (exclamation mark). The following table shows the key or key sequence and the NMODE command that provides some type of scrolling. The command name implies the functionality and you can get additional information in the *NMODE Command Reference* manual if necessary.

▼	Scroll Down command.
▲	Scroll Up command.
Prev	Scroll Pane Down Page command.
Next	Scroll Pane Up Page command.
C-X <	Scroll Pane Left command.
C-X >	Scroll Pane Right command.
C-M-V	Scroll Other Pane command.

As with many commands, it is often helpful to use **C-U** *integer* to give these commands prefixed arguments.

Writing and Editing Lisp Code

Writing New Code

Like writing and editing text, all Lisp code editing takes place in a buffer. You can create and enter a new buffer for editing Lisp with the Select Buffer Command, **C-X B**. If you want to create and begin editing a new Lisp source file, use the Find File Command, **C-X C-F**. If you named your new file with a `.l` suffix, you will automatically be put into Lisp mode. If you didn't, or you just created a new buffer (not a file), you may need to execute the Lisp Mode command, **M-X Lisp Mode**. Lisp mode provides some features designed for editing Lisp, like parenthesis matching and automatic indenting. Since these capabilities are available when editing new or existing code, they are discussed below under "Editing Existing Code."

Editing Existing Code

The next several subsections describe commands and techniques that access NMODE capabilities that are appropriate for editing Lisp.

Finding a File

In either a buffer or browser, you can access an existing file by executing the Find File command, **C-X C-F**, when you know the name of the file you want to examine. The command prompts for the name of the file, reads the file into a buffer it creates and then visits the buffer. You can edit the file as desired, using assorted search commands to find specific items in the file. This is the most direct way to get to a specific file.

You can use file name completion to find the file name. When the find file command prompts you for the file name, type a few letters of the file name and then press [ESC]. If the letters you typed are unique to a file name, the rest of the name is written for you. You press [Return] to accept the name. Or you can back space over the name and try again.

If you do not know the name of the file you wish to edit, you can use the directory facility to activate and then browse the directories you are interested in. This technique is also useful when you have several files you wish to edit that are all in the same directory.

Recall that a file in an active directory in `Directories` can be accessed in two main ways:

1. Execute the Find Directory or Dired command, **C-X D**, enter the name of a desired directory, and then point to and browse into a file; or

2. Execute **M-X nmode root**, which invokes the NMODE Root. Then, point to and browse `Directories`. From here, point to and browse into a desired directory, and then point to and browse into a desired file.

Finding a Specific Word in a File

The Incremental Search command, **C-S**, and the Reverse Search command, **C-R** let you find specific items in the current buffer (file). These commands were described in the previous chapter. Briefly, they prompt you for a string and begin searching for a matching string character by character. To go to the next occurrence of the search string, repeat the command. To terminate the search, type ESC.

The Word Search command, **C-M-S**, works in any language mode and prompts for entry of a word to find (a string). Note that a word is a string which has a blank or carriage return before it and a blank, carriage return, period or parenthesis after it. The word search command searches the current buffer from the location of the cursor forward to the end of the buffer, stopping just after finding the first occurrence of the word, or beeping the bell on not finding the word. While the command searches for a "word", you are not actually so restricted and can enter any reasonable number of characters; for example,

```
(cons 'foo bar)
```

Finding Function Definitions

The Find Item command, **M-.**, which is called the Meta-dot command in the tradition of Lisp editors, lets you find a function definition that might exist in any of the files that have been processed and added to a browser for code indexes! Recall that adding the files extracts the items and places them in the code index relative to the file that contains them.

The **M-.** command is only useful if the following conditions are fulfilled.

1. The code index facility has been loaded.
2. The file(s) containing the source code you are interested in have been added to a code index.
3. The appropriate code index is in memory (i.e. it has been browsed during the current session).

Once these conditions have been met, the **M-.** command works equally well in a code index or in an editing buffer.

Executing **M-.** (a period) prompts you to enter the name of an item, with the default being the item at the current cursor position; for example:

```
    Type in the name of the Item you want to see. (Default is: 'SETF')
```

If the default is not the item you wish to see the source for, enter the name of the function you want to find, which must be in some file in the browser for code indexes. On finding a match, the command creates and enters a buffer in Lisp mode with the associated file and with the cursor at the specified function.

In a browser, the command works the same way except that no default function is offered because the cursor is not next to any "item". You must enter the name of the function you want to find. In the browser for code indexes, just point to the index (function) and browse into it. This will find the desired code.

On failing, the command indicates that the item cannot be found.

Manipulating Lisp Code

For historical reasons, an expression at the top level in a buffer is called a **defun**. To increase efficiency, NMODE assumes that any (in column 0 initiates a **defun**; that is, any open parenthesis in column zero begins a top level expression. Assorted commands use this principle for locating defuns, and for indenting Lisp code based on the degree of nesting of expressions.

During an editing session, you need to focus on high-level tasks; for example, fixing code so it will work. To help you do this, Lisp mode in Emacs mode provides many commands for indenting code, setting marks, and moving the cursor, that are specifically suited to Lisp. For example, in Lisp mode, [Return] inserts a newline and indents to the appropriate place on the next line.

The commands that are described in the following subsections let you set specific marks, indent Lisp code, or move over Lisp code. The commands "know" how to handle comments, literal strings, and token syntax during execution and editing, provided that any literal strings do not extend over multiple lines.

These commands for working with Lisp source code are not isolated commands. They work in addition to other general commands for setting marks and moving over text. You are not limited to using "Lisp" commands. You can also use most of the commands discussed in an earlier chapter about writing and editing text.

Limitations

NMODE does not handle all the cases of parentheses and macros as they appear in Lisp code. In particular,

- Within a string that continues over a line boundary, parentheses and indentation may not match.
- Read macros (items beginning with #) are not handled properly.
- Indentation commands do not recognize all Lisp special forms and macros, nor do they recognize user defined macros.

These limitations should not pose a problem, however you should be aware of them. Before deleting a defun or region, check that what NMODE has determined to be the defun or region is what you want deleted.

Moving Over Source Code

Use the following commands to move over code in various increments.

C-M-[	The Move Backward Defun command moves the point to the location just before the beginning of the current defun. Alternately, you can execute **C-M-A**.
C-M-]	The Move Forward Defun command moves the point forward to the location just after the end of the current defun. Alternately, you can execute **C-M-E**.

C-M-F The Move Forward Form command moves the point forward over the contents of a form according to the "a-b-c-d" scheme below, but with some qualifiers. If possible, the command stays at the same level of parentheses. Otherwise, the command might move up or down a level, which is explained later. The scheme is: (a) If the first significant character after the point is an open parenthesis, (, the command moves the point past the matching close parenthesis,); (b) If the first character is a close parenthesis,), the point just moves past it; (c) If the character is part of an atom, the command moves the point to just after the end of the atom; or (d) With a prefixed argument, the command is repeated the specified number of times, forward with a positive argument and backward with a negative argument.

C-M-B The Move Backward Form command generally works like the Move Forward Form command, except in the opposite direction. In moving over a form, characters such as ' are also moved over. For example, when the point is after `'FOO`, executing **C-M-B** leaves the point before the single quote, not before the `F` in `FOO`.

C-M-N The Move Forward List command moves the point over a list somewhat like the related commands for forms, except that it does **not** stop on atoms. After moving past an atom, the command moves over the next expression, stopping after moving over a list. Thus, executing **C-M-N** lets you avoid stopping after each atomic item in an expression.

C-M-P The Move Backward List command works like the Move Forward List command, except it works in the opposite direction.

M-M Executing the Back To Indentation command moves the cursor forward or backward to the first nonblank character on the current line. If the cursor is to the left of the first non-blank character, **M-M** moves the cursor forward to that character. If the cursor is "in" the line and past the initial blanks, **M-M** moves the cursor back to the first non-blank character. Alternately, you can execute the same command with **C-M-M**.

Changing Levels of Lists

You might need to change levels while moving around in a large defun. The Forward Up List command, **C-M-)**, moves forward up past one closing parenthesis. The Backward Up List command, **C-M-(** or **C-M-U**, moves backward up past one open parenthesis. Both commands accept arguments, which repeat the command as specified by the argument. A slightly different command, the Down List command, **C-M-D**, moves down in list structure in about the same manner as searching for an open parenthesis, (, except that the command does not search past the end of the enclosing form.

Indenting Lisp Source Code

Along with the capability to move around in defuns, forms, and lists, you can also indent source code in assorted ways. The indentation can get scrambled during an editing session in which you find, move over, and edit the code extensively.

The traditional pattern for indentation uses the following scheme:

1. The open parenthesis, (, in the first line of a **defun** appears in column **0**. Open parenthesis for other expressions in the defun are indented. Here is an example:

   ```
   (defun junk ()
     (setq x '(1 2 3))
     )
   ```

   Notice that the (before `defun` begins in column **0** while the other expression, `(setq x '(1 2 3))` is indented.

2. The second line of a single expression is indented under the first element of that expression when that element is on the same line as the beginning of the expression; otherwise, the second line is indented two spaces more than the entire expression. Here is an example of the first case:

   ```
   (setq a
     (make-array 12))
   ```

   Notice that the second line, `(make-array 12)`, is indented under the first element, `setq`, of the initial expression and that the first element, `setq`, is on the same line as the beginning of the expression. Here is an example for the "otherwise" case:

   ```
   (defun junk (a) (setq
                     x a)
     )
   ```

   Notice that the second line, `x a)`, is indented two spaces more than the entire expression for it, which would have been `(setq x a)`.

3. Each subsequent line is indented under the previous line whose nesting depth is the same. You can see this in the above examples.

While there are few guidelines, and most software engineers have personal styles, most Lisp code is indented (pretty-printed) something like this:

```
(defun check-junk-values (a b c)
  (let ((x (+ (* a b)
              (* a c)
              (* b c)
              ))
        )
       (cond ((> x 104) 'too-big)
          ((= x 104) 'jackpot)
          ((< x 104) 'too-small)
          )
     )
  )
```

The following procedures, or commands, let you properly indent a single line, a specified set of lines, or all the lines in a form or defun.

C-M-I The Tab command, or Lisp Tab command, indents a line by calling the `lisp-tab-command` function. The Tab command indents according to the conventions for Lisp. In most cases, you press Tab or execute **C-M-I** at the beginning of a line. For a line that appears somewhere in a segment of code, position the cursor anywhere on the line and execute **C-M-I** to indent the line accordingly. In Lisp mode, the Tab command aligns lines according to the line's depth in parentheses relative to the line above it.

Return In Lisp mode, the Return key inserts a new line and moves the cursor down to the new line at the correct level of indentation. When you type Return at the end of a line, the command makes a following blank line and indents it appropriately. In the middle of a line, typing Return breaks the line and provides appropriate indentation in front of the new line.

C-M-Q The Lisp Indent Sexpr command indents the items in a single form when you locate the cursor under the form's open parenthesis, (. The initial line of the form is not reindented, only the relative indentation within the form is changed. Its position is not changed. To also correct the position, execute a Tab command, **C-M-I**, before you execute **C-M-Q**. This command is very handy for properly indenting Lisp code during extensive editing.

C-M- The Lisp Indent Region Command indents an entire region. Set a mark and then move the cursor (point) to create a desired region. Executing **C-M-\** applies the Tab command to each line whose first character is between the point and the mark; that is, in Lisp mode, the command does a Lisp indent on each line in the region.

M-^ To join lines, the Delete Indentation command is the inverse of the Newline command. Executing **M-^** deletes the indentation at the front of the current line, and joins it with the previous line, according to the following scheme: (1) Normally, the indentation and line separator are replaced by a single space; (2) If done before a), after a (, or at the beginning of a line, there is no space in the replacement; or (3) With a prefixed argument, **M-^** joins the current line and the next line, removing indentation at the front of the next line beforehand.

Additional Indentation Commands

Besides the commands just described, the following commands and techniques provide, alter, or adjust indentation.

M-\ Executing the Delete Horizontal Space command deletes spaces and tabs on either side of the point.

To insert an indented line before the current one, execute these commands in order:

1. Move To Line Start, **C-A**, which moves the cursor to the beginning of the line;
2. Open A Line, **C-O**, which opens a line above the current line; and
3. Execute either $\boxed{\text{Tab}}$ or **C-M-I**, which tabs over to the correct indentation level to begin the line.

To insert an indented line after the current one, execute these commands in order:

1. Move To Line End, **C-E**, which moves to the end of the line; and
2. Return, inserts a newline and indents the correct amount.

Matching Parentheses, Braces, and Brackets

Language modes automatically provide matching of parentheses, braces, and/or brackets. In Lisp mode, typing a close parenthesis) causes the cursor to move to the corresponding open parenthesis, pause briefly, and then move back to the inserted close parenthesis. This way, you can see whether expressions at all levels in your Lisp code are properly enclosed by parentheses. The point is not affected by matching operations. Only the cursor moves briefly. Therefore, you can type ahead freely as desired.

The following two commands help you insert and indent when you edit code, particularly when you are scanning code and see assorted places where you need to insert segments of Lisp code.

M-(The Insert Parenthesis command inserts a set of parentheses, (), at the cursor's location, which makes it handy to insert a Lisp expression between the parentheses and not worry about matching. With a positive integer command argument *n*, the command places the close parenthesis,), after the *n*th following expression.

M-) The Move Over Parenthesis command moves the cursor past the next close parenthesis and reindents, or when necessary, moves the remainder of the current line down and indents or opens an indented line.

In Lisp mode, matching occurs only for parentheses, (), not for brackets [], or braces, {}. In addition, the bell rings when you type a closing parenthesis that has no corresponding opening parenthesis.

A Brief Practice Session

If you are not familiar with the commands described so far that relate to manipulating source code (i.e. commands for setting marks, moving over source code, indenting, and matching), you can practice effectively by getting into a editing buffer in Lisp mode and entering a trivial defun such as the following:

```
(defun junk ()
  (let ((a 'nested-letters)
        (b '(a b c))
       )
    (list a (car b) (cdr b) (list b))
    )
  )
```

Using such a defun, go through the previous sections and practice with each command, observing cause and effect relations. You will quickly become proficient this way. Should you evaluate the form by executing **C-] E** or [Enter], the form should return the following value in the OUTPUT buffer:

```
JUNK
```

And if you evaluate

```
(junk)
```

the form should return:

```
(NESTED-LETTERS A (B C) ((A B C)))
```

Headers and Comments

Headers and comments are inserted in source code to explain that code. Headers (and file suffixes) are also important in NMODE because they can directly influence how an editing environment is configured. For example, certain commands for browsers use a file's header or suffix to determine its default environment. By knowing a file's type, NMODE can provide appropriate features in the buffers and browsers. Consequently, it is advantageous to initially include, and keep current the header at the beginning of a source file. In fact, NMODE does some of the work of updating the header for you. Other program comments should be included as required. The semicolon ; is the Common Lisp comment character.

NMODE provides the following commands for including headers and comments in source files.

M-X make header	Inserts a header template at the beginning of a file, appropriately enclosed by semicolons. The template contains several fields that you can edit; for example, author, language, package, and so on. Some of these fields will be filled in automatically based on the values of variables like `nmode:*author-name*`. If a source file exists and has no header, you can provide one by visiting or finding the file, executing **M-X make header**, regardless of the location of the cursor, and adding the information. This command also works in Text mode.
M-X make revision	Lets you insert a block of comments concerning a program revision just after the location of a header. The command automatically inserts comment characters at the left margins. You can execute this command with the cursor at any location in the source file.
M-X block comment	Inserts two lines of comment characters that are separated by a line which begins with one comment character. The cursor is placed just after the initial comment character, ready for you to type the comment. Note that the command does not insert a comment character at the beginning of any new line you type so you need to begin each new line with a comment character. This command works best for inserting high level comments as required in a source file.

M-; Moves the cursor to the end of the current line and inserts a comment character, placing the cursor just after the character ready for you to type a comment. This command works best for inserting short comments at the end of particular lines. No comment character is automatically inserted at the beginning of any subsequent lines.

M-Z Fills a paragraph of comments much like the filling of a paragraph, but without disturbing the format. Specifically, the command removes unnecessary spaces and gaps. Position the cursor anywhere inside the comment block and execute **M-Z**. Do not use the command inside a header, as it will delete the new lines which separate the fields.

Killing, Yanking, and Moving Lisp Code

The commands described earlier dealt with configuring an editing environment and performing fundamental editing tasks: indenting, inserting comments, matching parentheses, and so on. This section discusses ways to move source code from one location to another.

The functionality of some commands changes among language modes according to conventions used in the corresponding language. These changes are mentioned on a per command basis.

Killing and Yanking Source Code

Some of the following commands for killing, copying, and reinserting were discussed in the earlier chapter called "Writing and Editing Text". When used with the commands shown here for marking Lisp forms, they can also help you manipulate source code.

C-M-H The Mark Defun command establishes a defun as a region, putting a mark after its end and moving the point before its beginning (the cursor sits on the open parenthesis). Alternately, you can use **C-M-Backspace**.

C-M-@ The Mark Form command establishes a form as a region, but you need to be aware of the location of the point (cursor). The region marked is from the cursor position to the end of the form. The end of the form may be a word or a closing parenthesis. You can check the location of the mark with **C-X C-X**.

C-W The Kill Region command kills (removes) the current region. Before you execute the command, use an appropriate command to set a mark and move the cursor so that the mark and cursor establish a "killable" region.

M-W The Copy Regions command places a copy of the current region on the kill ring where it can be yanked. The copied region is not removed.

C-Y The Yank Last Kill command lets you reinsert killed source code after you kill the code and move the cursor to a location at which you want to reinsert the code.

M-Y The Unkill Previous command replaces reinserted killed text with the previously killed text, rotating the kill ring backwards.

Remember that you are not limited to the above commands. You can also use the general commands for killing characters, works, and lines when you edit Lisp code; for example, **M-D**, **M-Backspace**, and **C-K**.

The following commands let you kill specific units of Lisp code.

C-M-K The Kill Forward Form command kills forward over a form; that is, the command kills the characters that the Move Forward Form command, **C-M-F**, would move over.

C-M-DEL	The Kill Backward Form command kills backward over a form; that is, the command kills the characters that the Move Backward Form command, **C-M-B**, would move over.

Transposing Source Code

The commands for transposing characters, words, and lines in Text mode provide the same functionality for equivalent items in a Lisp mode.

C-T	Transposes characters.
M-T	Transposes words.
C-X C-T	Transposes lines.

C-M-T, the Transpose Forms command, drags the previous form across the current form, leaving the point at the end of the transposed forms. For example, given two forms such as:

```
(+ 2 3) (* 3 4)
```

with the cursor under the open parenthesis in `(* 3 4)`, executing **C-M-T** drags the `(+ 2 3)` across and the result looks like this:

```
(* 3 4) (+ 2 3)
```

with the cursor after the last close parenthesis. A positive argument serves as a repeat count, and a negative argument drags backwards, which in effect would cancel the effect of the command with an equivalent positive argument. An argument of zero transposes the forms at the point and the mark.

Beyond this, to effect any block move, you can establish a region, kill or copy it, move the cursor to a desired location for insertion, and yank it.

Executing and Debugging Code

NMODE provides assorted commands for interpreting, debugging, and compiling Lisp source code.

Commands related to evaluation and simple debugging of Lisp code are often prefixed with **Lisp**. For example, **Lisp** e evaluates a form. The **Lisp** prefix, is typed as C-]. Some of the lisp commands are bound to softkeys.

Evaluating Lisp Source Code

Another functionality related to the underlying Lisp language is that you can evaluate Lisp expressions directly in the NMODE environment.

Typically, you evaluate Lisp expressions by executing one of the following commands in a buffer in Lisp mode. Returned values are written to the OUTPUT buffer, which is automatically displayed if necessary.

C-] **E**	The Execute Form command passes text to the Lisp reader, which reads and evaluates the form starting at the beginning of the current line. The command sets a mark at the location of point and moves the point at the beginning of the line after the form. This way, you can easily return to the starting location by executing the Exchange Point and Mark command, **C-X C-X**. If two or more forms are on one line, the command evaluates the first form. Note that the given key sequence represents **Lisp** e. Alternately, you can press [Enter].
C-] **D**	The Execute Defun command passes text to the Lisp reader, which reads and evaluates the current defun. The cursor (point) can be anywhere in the defun. During execution, C-] **D** sets a mark at the current location of the point and moves the point to the beginning of the line after the defun, so you can return to the initial location. If there is no current defun, the Lisp reader reads and evaluates a form starting at the current location, which might not always make sense if the cursor is in some "intermediate" location between apparent forms. Note that the given key sequence represents **Lisp** d.

C-] . The Execute From Point command passes text from the current buffer starting at the point to the lisp reader. A mark is set at the point and the point is moved to the beginning of the line after the end of the form read.

Besides these commands, the Eval Form command, **C-X C-E**, provides a way to evaluate short forms regardless of what mode you are in. That is you can evaluate a form from any buffer or browser without modifying the current buffer. Executing **C-X C-E** prompts in the message area for entry of a form. Some forms you might typically execute this way include:

```
(require "lang-edit")
(in-package 'user)
(require "code-browser")
(* (+ 34 56 78)(+ 112 323 456))
```

Press Return after you type the form. The return value of the form will briefly appear in the echo area. Note that executing this command with a zero or negative command argument will not evaluate the typed-in form, but merely displays NIL.

Yanking Results from The OUTPUT Buffer

When you write Lisp code that serves as an example, and you want to include the returned values with the code, the Yank Last Output command, Lisp y or C-] **Y**, lets you insert the values returned during evaluation of Lisp forms or defuns, into the current buffer. Position the cursor at a desired location for insertion, typically just after a corresponding segment of code, and execute C-] **Y**. This yanks the last output to the OUTPUT buffer into the current buffer. This command is very useful when you write documentation for code and want to show the relationship between code and returned values.

Simple Debugging of Lisp Source

Lisp debugging is discussed extensively in the "Debugging Tools" chapter of the *LISP Programmer's Guide*. This section skims over a few simple things that you may encounter while running Lisp from NMODE. When an error occurs during Evaluation of a form, the system is put into a "debug", or "break" loop, which is indicated by a number in parentheses near the Lisp prompt at the right end of the OUTPUT buffer's mode line. For example, a message such as:

```
42 DEBUG (4)
```

indicates that you have evaluated 42 forms so far in NMODE and you are in the Debugger at the 4th break loop. Executing **C-] A** aborts all break loops and **C-] Q** quits the current break loop. If you abort all break loops, the OUTPUT buffer displays:

```
Abort to top loop!
```

You can continue to evaluate Lisp forms in a break loop. Additional errors invoke a new break loop and increment the break loop counter.

When you are in a break loop, the following commands let you exit a loop, or loops, or do some very simple debugging.

C-] ?	The Lisp Help command displays in the message area the commands available in a break loop. Note that this key sequence may also be referred to as **Lisp ?**.
C-] A	The Lisp Abort command pops out of any arbitrarily deep break loop and returns to the top level. The OUTPUT buffer's mode line reveals this by removing the indication that you are in a break loop. Note that this key sequence may also be referred to as **Lisp A**.
C-] Q	The Lisp Quit command exits the current break loop, moving up one level. The OUTPUT buffer's mode line confirms this by decrementing the break loop counter. Note that this key sequence may also be referred to as **Lisp Q**.

C-] B The Lisp Backtrace command enters the Execution Stack browser if it is loaded. This helps you see how an error occurred. The Execution Stack browser is discussed in detail in the "Debugging Tools" chapter of the *Lisp Programmer's Guide.* Note that this key sequence may also be referred to as **Lisp B**.

C-] C The Lisp Continue command attempts to continue execution from the point of the error. If the error is continuable, you may be prompted for information to correct the error. If the error is not continuable, a message to that effect will be printed in the OUTPUT buffer and you will remain in the current break loop. Note that this key sequence may also be referred to as **Lisp C**.

C-] L The Exit NMODE to Lisp command exits NMODE and invokes the Lisp Listener Loop. You can return to NMODE by executing `(nmode:nmode)`. This key sequence may also be referred to as **Lisp L**.

Compiling Lisp

The subject of compiling Lisp code is covered in the *LISP Programmer's Guide.* As an alternative to using the Lisp function `compile-file`, NMODE provides one command (**M-X compile file**) that compiles Lisp source code (the command is a front-end to `compile-file`).

To use this command, you must be in Lisp mode. Make sure that you have written the latest changes to the file you want to compile.

When you execute the Compile File command, **M-X compile file**, you will be prompted for the name of the source file you wish to compile (defaults to the current visited file), and for the name of the binary file to be created (defaults to the name of the current visited file with a `.b` suffix). Output messages from the compiler are written to the OUTPUT buffer.

Getting Additional Information about Programming

The information presented in this chapter can let you write, edit, and otherwise manipulate Lisp source code, but there is much more to the general task of programming. By using the information contained in the entire manual set for the Common Lisp development environment, you should be able to find needed information about programming at increasingly complex levels and across assorted categories.

Besides this chapter, the information contained in several earlier chapters that discuss facilities can help you work with code. In particular, code, search, and error indexes let you find code very efficiently, and you should learn how to use these facilities effectively and efficiently. While you might want or need to get started on a project within your company immediately, taking a few days to become familiar with the use of facilities for working with code and taking the time to "play" with some "dummy" code can save much time later on.

Establishing a Programming Environment

Editing Lisp code is not just a matter of typing text into a buffer. Since the editor is integral to the programming environment, you must make sure that it is configured correctly for the code you are editing and executing. Additionally, there are NMODE facilities that can make your programming task easier. This section covers some of the things that are useful (or necessary) when programming.

Using Packages

Packages are mappings between names and symbols. The same name can refer to two different symbols if the symbols are in different packages. Completing certain tasks often requires that you be in the correct package. To change the current package (the package that symbols are looked up in by default), use the **M-X set package** command or evaluate a form like the following ones.

```
(in-package 'user :use '(hp-ux_3w))

(in-package 'lisp)
```

You probably evaluated one of these forms in your initialization file. Common Lisp: The Language by Guy Steele, which accompanied your manual set, has information about packages.

If you wish to determine which package is active, execute **M-X show package**. The command briefly displays the name of the current package in the message area; For example,

```
The current package is USER
```

NOTES

Chapter 11
Working with Other Code

Introduction

Previous chapters described how to work with Conventional text and Lisp code. This chapter extends information in those chapters by discussing how to work with other code (i.e. C, Pascal, and Fortran). Note that this chapter describes the NMODE tools for **editing** non-Lisp source code. For a description of how to call non-Lisp routines from Lisp, see the chapter "Calling Non-Lisp Routines" in the *LISP Programmer's Guide.*

If you load the language module, the NMODE environment provides editing modes within Emacs mode for working with "other" code: C, Pascal, and Fortran. The following general ways to work with other code are available.

- You can browse, find, or visit a source file and then edit the file, using the typical commands for editing code.
- You can browse source code files in a way that produces an online index of source code items such as procedure and function declarations.
- Compiling source code produces a browser whose indexes point to compilation errors in a source file. After compilation, you can browse the errors to find offensive lines in the source code.
- Besides typical writing or editing, you can insert special templates into an editing buffer. The templates make it easier to write segments of code that can be subsequently inserted into a program.
- During editing, the language mode you are using provides block matching that is similar to the matching done for Lisp code, but is adjusted to suit the conventions of the current language.
- By adding appropriate headers and suffixes to other language files, you can help the system work for you. Many commands use a file's header or suffix to provide commands specifically suited to the current other language mode.

In general, if you are already skilled in the use of C, Pascal, or Fortran, you can work with those languages in the NMODE environment according to the conventions of the language and take advantage of the features just mentioned.

This chapter discusses the specific NMODE commands and procedures that are available for working with non-Lisp code. It does not re-examine more general commands or procedures for editing that were discussed in earlier chapters on text and Lisp.

Setting Up

As with text or Lisp, you must set up a working environment. The next few subsections discuss things you need to do to make available the commands that are specific to (non-Lisp) code.

Loading the Language Module

To work with and execute non-Lisp-specific commands, you need to load the code that implements them. This can be done in two different ways. The simplest way is to browse into Additional Facilities, put the cursor on the `Program Editing Support` item, and execute either of the `Load` or `Browse/execute` command line commands. You can also load the code index support functions by evaluating a Lisp form. If you expect to be editing non-Lisp programs often, you can have program editing support loaded automatically by adding this form to your initialization file (`.nmoderc` in your home directory).

```
    (require "lang-edit")
```

Invoking a Language Minor Mode

Depending on how you load a source file into an editing buffer, the buffer may have a Text, Lisp, C, Pascal, or Fortran minor mode.

The Default Minor Mode

A default language mode is established in your NMODE initialization file according to the value given to the global variable `nmode:nmode-default-language-mode`. The possible forms to set this variable are:

```
    (setf nmode:nmode-default-language-mode nmode:c-language-mode)
    (setf nmode:nmode-default-language-mode nmode:Lisp-language-mode)
    (setf nmode:nmode-default-language-mode nmode:pascal-language-mode)
    (setf nmode:nmode-default-language-mode nmode:fortran-language-mode)
```

If NMODE cannot determine the appropriate minor mode from either the header of a file or the suffix of the file's name, it will use the default minor mode.

Suffixes Invoke a Minor Mode

NMODE recognizes file suffixes such as `.c` or `.p` and can use the suffix (or the header) to automatically invoke a minor mode when you browse into or find a file. The file types are:

- The C language suffix is `c`; for example, `longpool.c`
- The Pascal language suffix is p; for example, `acebase.p`
- The Fortran language suffix is f; for example, `fast-calc.f`

A consistent use of suffixes can ensure that the "correct" minor mode is invoked when you browse or find an existing file.

Invoking a Minor Mode Directly

You can invoke a desired minor language mode directly in an editing buffer. Just execute **M-X** and enter one of the following strings:

```
C Mode
Pascal Mode
Fortran Mode
```

and press Return.

Always check the mode line to see if the buffer has the "right" mode; for example:

```
@ Emacs (C) $HOME/myproject/prog22.c  --24%--
```

indicates that you are in C mode with an associated file named `prog22.c` and the cursor is 24% of the "distance" from the top of the file.

Working With Other Code

The next several subsections discuss how to work with C, Pascal, or Fortran source code.

Moving the Cursor

A language mode has commands that move the cursor (point) forward or backward over a particular block of code. None of the block matching commands work in Fortran mode. Note below that the key sequences call the same commands with regard to name in both C and Pascal modes. But the commands attempt to move the cursor to the corresponding item in the source code according to the particular conventions for the language. The cursor need not be directly on an item, only between the corresponding items. The commands ignore string literals and comments. The system attempts to ring the bell when you type a closing character that has no corresponding opening character.

C-M-F	The Match Opening Keyword command moves the cursor forward to the next keyword. An alternative key sequence for this command is **M-{**.
C-M-B	The Match Closing Keyword command moves the cursor backward to the previous keyword. An alternative key sequence for this command is **M-}**.
C-M-N	The Match Opening Bracket command moves the cursor forward to the closing bracket that corresponds to the opening bracket. An alternative key sequence for this command is **M-(**.
C-M-P	The Match Closing Bracket command moves the cursor backward to the opening bracket that corresponds to the closing bracket. An alternative key sequence for this command is **M-)**.

In addition to these commands, the following matching occurs automatically when you type or edit code in the following language modes:

C mode	Matching occurs for parentheses, brackets, and braces.
Pascal mode	Matching occurs for parentheses and brackets.
Fortran mode	No matching occurs.

Use the same commands for indenting other language code as you used for Lisp code. The current language mode accommodates the commands and indents according to the conventions for the language.

Using Templates

Each language minor mode provides a set of predefined templates if you loaded the language module. A **template** is a portion of text that represents a construct in the language. A template contains certain **keywords**, which you do not need to retype, that appear in lowercase letters. The template also contains **placeholders** that appear in uppercase letters. You replace the placeholders and enter code according to what you want a program to do. For example, a Pascal **IF** template looks like this:

```
if EXPR then
begin
   STMT;
end;
```

where **EXPR** and **STMT** are placeholders you would replace with particular code.

In working with templates, you can use the following commands as desired:

- The Insert Template command, **M-O**, prompts for entry of the name of a template. Name completion works. Typing a question mark during entry displays names that match the partial entry. The template is inserted in the current buffer at the current cursor location and the cursor is placed at the first placeholder in the template. Typing any character inserts the character and deletes all the characters of the placeholder. Executing **M-X delete placeholder** lets you delete or insert any character in the current placeholder. Within a template:
  - Executing the Next Placeholder command, **M-N** moves to the next placeholder, and
  - Executing the Previous Placeholder command, **M-P**, lets you move among placeholders. If no next or previous placeholders exist, the system beeps and indicates this state in the message area.
- Executing **M-X edit template** lets you edit templates. The command prompts for entry of a template name, then invokes a buffer in which the template can be edited. While editing the template, you have available the following special editing commands.
  - Executing **M-X make placeholder** command lets you enter a string that represents a placeholder, which is inserted at the cursor's location.
  - Executing the Quit Template Edit command, **C-X C-S** or **C-X C-W**, creates a new template, overwriting any previous template with the same name and exiting the buffer. You must execute this command to have the new version of the template available. The edited template definitions are stored in the file `usr-tmplates.l` in your home directory.

- The **M-X Make Template** command is available for creating new templates. The command prompts for the name of a new template, then invokes a buffer in which the template can be created.
  - Executing the **M-X Abort Template Edit** command lets you discard any changes you have made to the template being edited. If you were editing an existing template, it is not overwritten. If you were making a new template, it is not created.

Available Templates

The following subsections list the templates and commands that are available in a particular minor mode when you execute the **M-X edit template** or **M-X insert template** commands. Remember that if you type ? at any time while you are entering the name of a template, you will be presented with a list of the templates whose names match the characters you have typed so far.

C Mode Templates

C mode provides the following templates.

```
block comment
case
comment
conditional
do while
for
function
header
if
if else
main
switch
while
```

No descriptions of the templates are provided because the names are self-explanatory.

Pascal Mode Templates

Pascal mode provides the following list of templates.

```
block comment
case
comment
for
function
header
if
if else
procedure
```

```
program
repeat
while
with
```

Fortran Mode Templates

Fortran mode provides the following list of templates.

```
arithmetic if
block comment
block if
comment
do
do while
else if
goto
header
logical if
parameter
program
```

Indentation

Nmode provides automatic indentation appropriate to each language when RETURN (or **C-J**) is used to terminate a line. In Pascal and Fortran, you can customize when indentation will occur by modifying the editing User Options. In C, indentation occurs when "{" is the last character on the previous line. You can also change the number of spaces to indent. In addition, the following commands relating to indentation are available:

C-M-I	The **M-X C, Pascal or Fortran Tab** command moves the current line to the appropriate indentation.
M-I	The **M-X Move Forward Indent** command moves current line forward the number of spaces for default indentation.
M-H	The **M-X Move Backward Indent** command moves current line backward the number of spaces for default indentation.

Editing Options

Each language (C, Pascal and Fortran) provides user customizable editing options. These options browsers are accessed by browsing the User Options facility from Nmode Root. You will see the items C Edit Options, Pascal Edit Options and Fortran Edit Options. Each browser provides the following menu of commands:

```
Help Browse/modify Group Filter Sort Write Restore-default Quit
```

What you will see on the screen when you browse any of the editing User Options items is described below. The default value for each item is indicated.

```
User Options:  C Edit Options
  Currently saved in "$LISP/config/c-edit.opt"

 C EDITING OPTIONS:
   Block begin character(s)                "{"
   Block end character(s)                  "}"
   Bounce Cursor with '}'                  Yes
   Spaces to Indent                        3
```

The first two items, Block begin and end character(s), allow you to tell Nmode if you have redefined their value as is allowed in the C language. Bounce cursor indicates whether you want to enable the feature that moves the cursor to the opening bracket and back, when a closing bracket is typed. Spaces to Indent gives the number of spaces to insert when indentation is done.

```
User Options:  Pascal Edit Options
  Currently saved in "$LISP/config/pscl-edit.opt"

 INDENTATION OPTIONS:
   Indent after Reserved Words      ("BEGIN" "CONST" "DO" "THEN" "ELSE"
                                     "FUNCTION" "OF" "PROCEDURE" "RECORD"
                                     "REPEAT" "TYPE" "VAR")
   Spaces to Indent                 3
```

Indent after Reserved Words provides a method of customizing the indentation algorithm. Indentation will occur when any of the keywords in the list is the last piece of text on the preceding line. You may add to or remove keywords from this list. Spaces to Indent gives the number of spaces to insert when indentation is done.

```
User Options:  Fortran Edit Options
  Currently saved in "$LISP/config/ftn-edit.opt"

 INDENTATION OPTIONS:
   Indent after Reserved Words         ("DO" "IF" "ELSE" "ELSE IF" "WHILE")
   Spaces to Indent                    3
```

Indent after Reserved Words provides a method of customizing the indentation algorithm. Indentation will occur when any of the keywords in the list is the last piece of text on the preceding line. You may add to or remove keywords from this list. Spaces to Indent gives the number of spaces to insert when indentation is done.

Compiling Other Code

To use NMODE to compile non-Lisp code (via the regular HP-UX compilers), you must have loaded the Compilation Error Index Facility (see Chapter 11). Once that is loaded, the **M-X Compile File** command works with non-Lisp code as long as you are in the appropriate language mode; for example, Pascal mode to compile Pascal source code.

Use of Facilities

When you work with a language mode, you can use all the NMODE facilities and in particular, you can make good use of code and error indexes. These facilities are described in earlier chapters.

Chapter 12
HP-UX Access Facility

Introduction

Being able to communicate "directly" with HP-UX while remaining in an NMODE buffer can be very useful. You can use NMODE's editing features to manipulate your HP-UX commands and responses. You can also use HP-UX tools to manipulate NMODE's data.

The HP-UX Access facility provides two ways to communicate with the underlying HP-UX operating system.

- The **System Shell** lets you execute an HP-UX command, or sequence of commands, from any NMODE buffer.
- A **shell-buffer** provides an NMODE buffer that emulates an HP-UX shell. The shell-buffer responds very similarly to a "dumb" terminal running an interactive shell.

Even if you choose not to load this facility, the HP-UX Execute Command (available on the [f2] softkey) lets you execute an HP-UX command from within NMODE (with some restrictions).

Loading the Facility

As with most pieces of NMODE, there are several ways to load the HP-UX Access facility. From the NMODE root, you can point to and browse `Additional Facilities` and then point to and browse `HP-UX Access`.

If the facility has not already been loaded, NMODE will ask:

```
Facility not yet loaded, do you wish it to be? (Default is: 'Yes')
```

Press [Return] to load the facility. Once the facility is loaded, you will be prompted:

```
Do you want to create default HP-UX shells? (Default is 'Yes')
```

Press [Return] to create the default shells.

As the shells are created, messages to that effect will appear in the message area. You will then be left in a shell-buffer.

Once the access facility has been loaded, repeated browsing of the `HP-UX Access` facility will provide the opportunity for creating more shells and modifying the options for this facility. The options will be explained in a later section of this chapter, as will other methods of creating shells.

At this point you are in an NMODE buffer that communicates with an HP-UX shell. Depending upon the setting of the options, you will have either a Bourne shell prompt or a C-shell prompt appearing on the left edge of the buffer. What you type is sent to HP-UX and the results are sent back to the buffer. Try typing an HP-UX command just as you would in a "regular" shell. For example, type `pwd` and press [Return]. The system will respond by printing your working directory. You can leave the buffer by executing a **C-M-L**.

Loading During Initialization

You may wish to have the HP-UX Access facility loaded during NMODE initialization. The form necessary to load the facility appears in both the `$LISP/config/nmode-init` "group-customization" script and your `$HOME/.nmoderc` "personal-customization" script. Although the form appears in both scripts, it appears as a comment. By editing your `.nmoderc` script, and removing the comment character from the beginning of the line containing the form, the facility will be loaded each time you use NMODE. Look for the form: `(require "shell-access")`. You should only uncomment one of the two occurrences of the form.

Alternate Loading

If for some reason you wish to load the facility "on-demand" from the keyboard, you can use the `EVAL-FORM-COMMAND` which is usually bound to **C-X C-E**. For example, typing: [CTRL]-[X] [CTRL]-[E] `(require "shell-access")` [Return] would load the access facility but not create the default shells. See the commands later in this chapter to create shells.

Comparison of Shells

The HP-UX Access facility allows up to ten shells to be in use at any given time. Thus, if you choose to have a System Shell, you may create up to nine shell-buffers.

The advantage of using a **shell-buffer** is that, in a shell-buffer, you can directly execute many HP-UX commands as if you were typing on a terminal to an HP-UX shell, **and** you can execute the NMODE commands available in an editing buffer in Emacs mode with Text and HP-UX minor modes. Also, shell-buffers are appropriate for executing "interactive" commands (those that expect user input) while the System Shell is not.

On the other hand, using the **System Shell** lets you execute HP-UX commands in **any** buffer, but you need to type the HP-UX command or commands and then execute the appropriate Emacs command (for example, **M-X HP-UX Execute Line** or **M-H E**). The output generated by the HP-UX command can be directed to any buffer.

Differences with Regular Shells

While interacting with a shell-buffer is similar to interacting with an HP-UX shell, note the following differences:

- Output often appears a batch at a time rather than a line at a time.
- Since the buffer collects all text, you can scroll back through all commands and responses since the beginning of the session.
- Certain HP-UX shell commands and applications that insist on being connected to terminal that is "smarter" than the **dumb** terminal type used by the facility are not likely to work as you might expect.
- No preprocessing or postprocessing is performed by the facility. For instance, some screen-oriented programs will not work as you expect; for example, vi will use "open" mode. Also, escape-code sequences present in your shell's prompt will not be detected.
- If you change your prompt, be sure to set the "prompt character" option. From the shell-buffer's point of view, the prompt and your commands are simply text. Usually, marks are used to delimit your commands so that NMODE knows what to send the shell process. Setting the prompt character option will allow NMODE to know what the prompt looks like and not pay attention to marks, allowing you to use marks as desired.
- When using a shell-buffer without the prompt character option, setting marks is likely to cause problems. The facility sets the marks with each command you type.

- If you use the *csh* history mechanism, be aware that several HP-UX commands may be issued for one NMODE/HP-UX command. Do not expect the last command in the history to be your last command. This makes the history event number less useful to include within your prompt.
- The facility uses a subset of the C-shell. HP extensions to the standard C-shell are not available.

Be sure to see the last section of this chapter for other anomalies.

Using the HP-UX Access Facility

This section describes the basic use of the HP-UX Access facility. The examples in this chapter assume you have loaded the facility and are using the default shells. The facility does allow you to create and kill shells at will and allows you to tailor them to your needs. Please use the default values until you become familiar with the features of the facility. Setting shell options will be discussed later in this chapter. The last section of this chapter has information to help you fix problems that may occur when you use this facility.

Using a Shell-Buffer

After loading the facility and creating the default shells, you will find a new buffer named `HP-UX.SHELL` has been added to the list of buffers. To see this, you can execute **C-X C-B**. You should see something similar to this:

```
Buffers

    Buffer Name           Size     File Name

S*  HP-UX.SHELL              1
    MAIN                     1
    OUTPUT                   8



    Browser (Buffers)
```

Note the meaning of the `S*` that appears to the left of the buffer name. The `S` indicates that the buffer is a shell-buffer. The `*` indicates that the buffer has been modified.

Point to and browse the HP-UX.SHELL buffer. The buffer may be empty except for the shell's prompt character. HP-UX will respond very much as it would in an "ordinary" shell. However, when a command produces several lines of output, the lines will often appear in the buffer all at once or in bursts.

On browsing into the buffer, you will see your prompt. You can now type in HP-UX commands. After echoing the command you typed, your input is sent to the shell-buffer's shell for processing and output is appended to your current buffer. For example, if you executed *pwd*, the display might look like this:

```
$ pwd
/users/joe
$
```

Within this shell-buffer, you can also execute the NMODE commands available in Emacs major mode with Text and HP-UX minor modes. The HP-UX minor mode commands are described in the next section.

For example, you can use the [↖] ("Home" key) to return to the beginning of your buffer, use **M-F** (move forward word) as many times as necessary to skip over your prompt, and then press [Shift]-[Return] (or **M-H N**) to re-execute the *pwd* command. Your output will again appear at the end of the buffer.

It is worth noting that the [Return] key has a special binding in a shell-buffer (**HP-UX Execute Line**). This binding is what gives the appearance that you are in a shell.

Filename Completion

As you type HP-UX commands, you might want to complete filenames just as you would in *csh*. The shell-buffer mechanism provides the HP-UX Escape command which attempts to complete filenames. To use it, you can press the [ESC] at any point in typing a filename and the command will complete the filename if possible. Alternatives for executing the command include: pressing the softkey for **Complete Filename**; clicking on the popup menu item called `Complete Filename`; and executing **M-X Complete Filename.**

The escape command follows NMODE's rules for filename completion and not *csh*'s rules. For example, the completion of a directory name will include a trailing slash (/). This feature is **not** available with the System Shell.

Using the HP-UX System Shell

If you have created the System Shell and are in **any** editing buffer (Emacs mode), you can execute the current line as an HP-UX command. The procedure is to type an HP-UX command and then execute an NMODE/HP-UX command such as HP-UX Execute Line. The command then submits your input to the System Shell, which interacts with the underlying HP-UX operating system. Results are appended to the OUTPUT buffer or to an alternate buffer you designate.

Do **not** change the System Shell's terminal characteristics, its environment variables (such as its prompt), or set variables such as autologout. It's best to leave it alone and let it work for you.

Redirecting Output

When using the system shell, results appear in the OUTPUT buffer unless you specify a buffer that is to receive HP-UX output. This is accomplished by executing the HP-UX Set Output Buffer command, **(M-H O)**. Note that this chapter uses the term "HP-UX Output Buffer" to mean either the OUTPUT buffer or the buffer you specify to recieve HP-UX Access output.

When using a shell-buffer, results are appended to the buffer and cannot be redirected. You can use the HP-UX Yank Last Output command to retrieve the results and put them in another buffer.

Note that using the HP-UX Filter Region command will redirect output to the buffer in which the command was executed. (Filters remove the input text and replace it with the output text.) You may wish to issue another HP-UX Set Output Buffer command after using the HP-UX Filter Region command.

Executing NMODE/HP-UX Commands

This section contains the NMODE commands that are used to interact with HP-UX. All commands execept the **HP-UX Set Output Buffer** command can be used in a shell-buffer.

The results (output) of a command executed in a shell-buffer are appended to the end of the shell-buffer. The results of a command executed in any other buffer are appended the HP-UX Output buffer.

There is one special command that is available with or without loading the HP-UX Access facility, it is discussed first.

HP-UX Execute Command

HP-UX Execute Command lets you execute an HP-UX command from **any** buffer or browser in the NMODE environment. This command's behavior depends upon whether or not the HP-UX Access facility has been loaded.

- If you have not loaded the HP-UX Access Facility, you can execute an HP-UX command by pressing the softkey for **HP-UX Command**. Output appends to the window from which you invoked NMODE and you cannot save this output to a file or manipulate it using Emacs. If you choose not to load the facility, be sure not to "bury" the window from which you invoked NMODE.
- Loading the facility and creating a System Shell allows you to use **C-X H**, **M-X HP-UX Execute Command**, or the **HP-UX Command** softkey to execute the command. Note that the same softkey is used with or without the facility, but that its definition has changed: output is appended to the HP-UX Output buffer. Be aware that since output is appended to a buffer that might not be displayed, you might need to have NMODE display the HP-UX Output buffer by selecting it or by any other means.

On executing the command, you will see this prompt:

```
Enter HP-UX command(s) to execute:
```

Enter a command, `ls /users` for example, and press [Return]. Alternately, you can enter a series of commands; for example:

```
pwd; ls; ps -ef
```

Either way, the command appends output according to the conditions mentioned above.

Note that "interactive" commands (like *more*) should not be executed in this fashion, since user input is required all at once.

General Commands

The following commands are available everywhere but in **input-mode**. You can use them while in any buffer or browser.

Shell Manipulation Commands

The general facility commands listed below are used to create and kill shells and to invoke the options browser interface to the access facility.

```
 Shell Allocation Commands            Shell Deallocation Commands
 -------------------------            ---------------------------

  Create HP-UX Shell Set               Kill All HP-UX Shells
  Create Default HP-UX shells

  Create System Shell                  Kill System Shell
  Create Default System Shell

  Create Shell-Buffer                  Kill Shell-Buffer
  Create Default Shell-Buffer          Kill Current Shell-Buffer


 Shell Recovery Command               HP-UX Access Options Command
 ----------------------               ----------------------------

  Recover shell-buffer Shell           HP-UX Access Options
```

Detailed explainations of these commands are provided later in this chapter.

HP-UX Execute Buffer

This command executes the entire buffer as input text to the shell. All input is executed in the shell's current environment, hence it can affect subsequent shell commands.

`HP-UX-execute-buffer-command`	**M-H B**
	M-X HP-UX Execute Buffer

HP-UX Script Buffer

This command executes the entire buffer as a **shell-script** (see *sh* or *csh* in the *HP-UX Command Reference* for information on shell-scripts). A sub-shell is spawned for execution of the script. Thus, any subsequent commands are not affected by the contents of the script.

Note that this command is equivalent to the **HP-UX Send Buffer** command with a shell specified as the HP-UX command.

`HP-UX-script-buffer-command`	**M-H M-B**
	M-X HP-UX Script Buffer
	Script Buffer popup menu item in a shell-buffer

HP-UX Send Buffer

This command sends the entire buffer as input text to the specified HP-UX command. You are prompted for the HP-UX command, and a sub-shell is spawned to execute it using the buffer as stdin.

`HP-UX-send-buffer-command`	**M-H C-B**
	M-X HP-UX Send Buffer

HP-UX Set Output Buffer

This command does not apply to shell-buffers. You are prompted for the name of a buffer that is to be designated the HP-UX Output buffer. The specified buffer receives output until you specify another buffer or end the current NMODE session.

Note that specifying a new HP-UX Output buffer takes effect immediately. If the System Shell is writing output to the HP-UX Output buffer and you specify a new HP-UX Output buffer before the shell has completed its command, part of the output will appear in the old buffer and part will appear in the new buffer.

`HP-UX-set-output-buffer-command`	**M-H O**
	M-H C-O
	M-X HP-UX Set Output Buffer
	Set Output Buffer popup menu item

Emacs Commands

The following commands can be used in any buffer, including shell-buffers. The only requirement is that the buffer is in Emacs mode. Additional commands that only work with shell-buffers are described after these commands.

HP-UX Execute Line

This command takes the line containing the cursor, "trims off the prompt", and sends the line to a shell for execution. Either NMODE will trim off the prompt by knowing the *prompt-end-character* because you have specified it in the options browser, or it will try to send the characters existing from the location of the mark to the end of the line.

`HP-UX-execute-line-command`	**M-H E**
	M-H C-E
	M-X HP-UX Execute Line
	Execute Line popup menu item

Additional bindings when using a shell-buffer:

`HP-UX-execute-line-command`	Return
	Execute Line softkey

HP-UX Execute Region

This command is similar to HP-UX Execute Buffer, but applies only to the current region. It takes a region of text and sends it to the current shell for execution. Commands executed in this way can change your environment.

`HP-UX-execute-region-command`	**M-H R**
	M-H C-R
	M-X HP-UX Execute Region

Additional bindings when using a shell-buffer:

`HP-UX-execute-region-command`	**Execute Region** softkey in a shell-buffer
	Execute Region popup menu item in a shell-buffer

HP-UX Script Region

This command is similar to the HP-UX Script Buffer command, but applies only to the current region. It takes the current region and sends it to a sub-shell for execution as if it were a shell script. This means your shell will not be changed by the execution of the commands within the region.

`HP-UX-script-region-command`	**M-H X**
	M-H C-X
	M-X HP-UX Script Region

Additional bindings when using a shell-buffer:

`HP-UX-script-region-command`	**Script Region** popup menu item

HP-UX Send Region

This command is similar to the HP-UX Send Buffer command, but applies only to the current region. It prompts you to send the region to a particular HP-UX command. The current region is then sent to the HP-UX command (a sub-shell) as that command's stdin.

`HP-UX-send-region-command`	**M-H S**
	M-X HP-UX Send Region

Additional bindings when using a shell-buffer:

`HP-UX-send-region-command`	**Send Region** popup menu item

HP-UX Filter Region

This command takes the current region, deletes it from the buffer, and replaces it with its filtered result. You are prompted for an HP-UX command such as *sort*, *cat*, *nroff*, etc. to apply as the filter.

`HP-UX-filter-region-command`	**M-H F**
	M-H C-F
	M-X HP-UX Filter Region

Additional bindings when using a shell-buffer:

`HP-UX-filter-region-command`	**Filter Region** popup menu item

HP-UX Yank Last Output

Executing this command yanks the last output from an HP-UX command, inserting it in the current buffer after the cursor (point). The last output could contain any amount of information from the most recently active shell that had any output, hence it will not always work as expected.

`HP-UX-yank-last-output-command`	**M-H Y**
	M-H C-Y
	M-X HP-UX Yank Last Output

HP-UX Send End-of-File

Use this command to send to the shell the character the terminal is currently using as the end-of-file (EOF) character. The default character is control-D. See *terminfo(4)* for details about changing the EOF character.

`HP-UX-send-interrupt-command`	**M-H C-D**
	M-X HP-UX Send Eof

HP-UX Send Interrupt

Use this command to send an interrupt to the shell. This is equivalent to a `kill -2` in HP-UX.

`HP-UX-send-interrupt-command`	**M-H C-C**
	M-X HP-UX Send Interrupt
	Send Interrupt popup menu item

Additional bindings when using a shell-buffer:

`HP-UX-send-interrupt-command`	DEL
	C-C

HP-UX Stop Output

Use this command to stop output from a shell.

`HP-UX-send-stop-output`	**M-H C-S**
	M-X HP-UX Stop Output

Additional bindings when using a shell-buffer:

`HP-UX-send-stop-output`	$\boxed{\text{Stop}}$

HP-UX Start Output
Use this command to resume output from a shell that was previously stopped.

`HP-UX-send-start-output`	**M-H C-Q** **M-X HP-UX Start Output**

Additional bindings when using a shell-buffer:

`HP-UX-send-start-output`	$\boxed{\text{Shift}}$-$\boxed{\text{Stop}}$

HP-UX Send Signal
Use this command to send a signal to the shell. You will be prompted for the HP-UX signal number. See the *kill(1)* and the *signal(2)* commands in the *HP-UX Reference* for details on signals.

`HP-UX-send-signal-command`	**M-H K** **M-H C-K** **M-X HP-UX Send Signal**

Commands Specific to Shell-Buffers
These commands can be used only in shell-buffers. All commands in the previous sections are available in shell-buffers except the **HP-UX Set Output Buffer** command.

HP-UX Execute to End
This command takes characters from the cursor to the end of the line and sends this line to a shell for execution. It is similar to HP-UX Execute Line, but does not trim the prompt. Hencs, you can always use this command to move to the correct position in a buffer and execute the portion of the line you desire.

`HP-UX-execute-to-end-command`	**M-H N** **M-H C-N** **M-X HP-UX Execute to end** $\boxed{\text{Shift}}$-$\boxed{\text{Return}}$ $\boxed{\text{Enter}}$ on the numeric keypad **Execute To End** softkey **Execute To End** popup menu item

HP-UX Escape

This command completes a filename if possible. See the earlier section in this chapter for details on filename completion.

`HP-UX-escape-command`	ESC key
	M-X HP-UX Complete Filename
	Complete Filename softkey
	Complete Filename popup menu item

HP-UX Send Character

This command is used to send a character to HP-UX commands that expect a single-character response not followed by a **newline**). For example, the *more* command wants a space character to scroll the next screenful of text.

`HP-UX-send-character-command`	**M-H A**
	M-H C-A
	CTRL-Return
	M-X HP-UX Send Character

HP-UX Execute and Delete

This command sends the current line (via HP-UX Execute Line) to HP-UX and then deletes it from the buffer. This can be useful for protecting passwords. Even though the password is printed as you type it, all referrences are obliterated as soon as you execute this command.

`HP-UX-execute-and-delete-command`	CTRL-**Meta**-Return
	M-X HP-UX Execute and Delete

The Options Browser

The easiest way to create or kill shells is to use the options interface to the HP-UX Access facility. To enter the options browser, you may either:

- Go to NMODE root (**C-X R**). Point to and browse the `User Options` then point to and browse `HP-UX Access Options`.
- Execute the **M-X HP-UX Access Options** command.
- Use the directory browser or the Find File command to browse the file: `$LISP/config/hp-ux.opt`

The options browser display looks something like this:

```
   User Options:  HP-UX Access Options
     Currently saved in "$LISP/config/hp-ux.opt"

  CREATION OPTIONS:
    Master Pty Directory                        "/dev/ptym"
    Slave Pty Directory                         "/dev/pty"
    Shell-buffer to create (name)               "HP-UX.SHELL-1"
    Unique character to identify prompt         NIL
    Type of HP-UX shell                         NIL
    Create System Shell and shell-buffer        <Browse to invoke function>
    Create System Shell only                    <Browse to invoke function>
    Create new shell-buffer only                <Browse to invoke function>

  KILL OPTIONS:
    Kill System Shell                           <Browse to invoke function>
    Kill shell-buffer                           <Browse to invoke function>
    Kill all HP-UX shells                       <Browse to invoke function>

  POPUP-MENU OPTIONS:
    Options for 'ps' command                    "-ef"
    Options for 'ls' command                    "-aF"
    Mailer to use for 'mail' command            "mailx"


 I/O @ Browser (User Options) HP-UX Access Options
 Help  Browse/modify  Group  Filter  Sort  Write  Restore-default  Quit
```

The three main sections are creating, killing, and popup options.

Creation Options

Use these options to create the System Shell or shell-buffers.

Master and Slave Pty Directories

Do not alter master pty and slave pty directories unless you are familiar with HP-UX device files. The master and slave pty directories should be the absolute pathnames where pty device files can be found. These options relate to the *pty(4)* driver, which provides a communication path between a supporting server process (the master side of *pty*, which is the NMODE Shell) and an HP-UX application process (the slave side of *pty*, which is sh/csh). See *pty(4)* in the *HP-UX Reference* for more information on ptys.

Shell-Buffer Name

This option allows you to change the default name for shell-buffers. As successive shells are created, a numeral will be appended to the name. The default creation functions will use the name specified here for the next shell-buffer.

Unique Prompt Character

As a typing-aid, so that you can use marks freely, you should notify NMODE of your prompt's ending character. Be aware that the value of *prompt-end-character* is only the ending character of your prompt. The character must be unique in the prompt.

A couple of examples should help to clarify specification of prompt-character option values:

HP-UX Prompt	**Prompt-Character Option**
"$ "	$ or <space> (See Note 1)
"%"	%
"% "	% or <space>
"`joe's system `"	`m` (See Note 2)
"`joe's sys`"	None Available (See Note 3)
"%%"	None Available (See Note 3)

Note 1

Since a blank preceding a command name does not cause any problems, the $ or the <space> character can be used as the unique end character.

Note 2
Since there are two blanks in the prompt "`joe's system `" (and therefore not unique) the `m` must be used. This works for the same reasons as discussed in Note 1.

Note 3
In both of these cases, a unique character at the end of the prompt (or at the end followed by whitespace) is not available. The option must be `nil` in these cases.

Type of Shell

Depending on your preferences, you may choose either a Bourne shell (`sh`) or a C shell (`csh`) as the HP-UX shell associated with the NMODE shell-buffers you create.

The default of `nil` indicates that NMODE should use the shell type defined by the shell variable `$SHELL` or, if the variable is not set, `/bin/sh` will be used.

Create Shells

Browsing these options results in the creation of the requested shells. See the later section on Creating and Killing Shells for more information.

Kill Options

Use these options to get rid of the System Shell or shell-buffers. See the later section on Creating and Killing Shells for more information.

Popup-Menu Options

These options allow you to customize frequently used HP-UX commands and arguments that will be available when you are using the HP-UX Access facility popup menus.

Options for *ps*

The HP-UX *ps* command provides process status information about your system. Browsing this option allows you to enter a new argument string for the *ps* command. See the *HP-UX Reference Volume 1* for other possible arguments.

Options for *ls*

The HP-UX *ls* command lists the files in your directory. Browsing this option allows you to enter a new argument for the *ls* command. See the *HP-UX Reference* for other possible arguments.

Options for Mailer

This option lets you specify the HP-UX command to be used for the mail command available through the NMODE/HP-UX popup menus. The default is *mailx.* For more information about mailers, see *HP-UX Concepts and Tutorials.*

Popup Menu Access

If you have an mouse and want to execute NMODE/HP-UX commands, clicking the right button when you are in a buffer will provide a menu with an Emacs menu as one of the items. For example, if you are in an Emacs buffer in Text minor mode you will see the following menu.

```
TEXT MODE

Yank
Undo
Kill >>
Jump >>
Mark >>
Transpose >>
Format >>
Auto Fill Toggle >>
Emacs >>
Nmode General >>
Help >>
Quit
```

Selecting the `Emacs` item results in the following menu:

```
EMACS

Modes >>
Places >>
Utilities >>
HP-UX Access >>
Nmode General >>
Help >>
Quit
```

Selecting the HP-UX Access item results in the following menu:

```
HP-UX ACCESS

Create/Kill >>
Set Output Buffer
Execute Line
Send Interrupt
Script Buffer
Process Status
Date
Mail
Show Current Directory
List Directory
HP-UX Ref Manual
```

The command called `Create/Kill` >> provides the following popup menu.

```
CREATE/KILL

CREATE
 System Shell & Shell-Buffer
 Shell-Buffer
 System Shell
KILL
 System Shell
 Shell-Buffer
 All Shells
```

These commands are discussed in the section on creating and killing shells.

The last six commands in the menu entitled "HP-UX Access" are commonly used HP-UX commands that NMODE provides an easy way to access. Some of these can be modified using the Options Browser for HP-UX Access. The remainder of the commands in the menu are discussed in the "Executing NMODE/HP-UX Commands" section of this chapter.

In addition to these menus (accessedvia Emacs mode), NMODE provides a set of menus pertaining to HP-UX minor mode, which are accessible via shell-buffers. Aside from adding some commands unique to shell-buffers, these menus are primarily present to give you faster access to NMODE/HP-UX commands. The commands accessible with this menu set will not be discussed further, as they are discussed elsewhere throughout this chapter.

Creating and Killing Shells

This section describes the various commands for creating and killing the System Shell and shell-buffers. The commands are discussed in terms of the **M-X** interface since all other interfaces present some subset of this functionality. All commands that include the word "default" imply that the HP-UX Access options are used as parameter values instead of prompting you for them.

Create Default HP-UX Shells

If the access facility is loaded and no system shell is currently allocated, executing **M-X Create Default Hp-Ux Shells** allocates the System Shell and a shell-buffer shell using the values specified in the Options browser. This includes the name, shell type, and *prompt-end-character* If none have been specified, the shell-buffer will be named `HP-UX.SHELL`, the shell type will be as specified in the Options section of this chapter with no *prompt-end-character*. If the System Shell already exists, just a shell-buffer will be created.

This command is also available through the Options browser and popup menu interfaces.

Create HP-UX Shell Set

This command differs from `Create Default HP-UX Shells Only` because it lets you specify parameters related to the shells. You are asked to provide a name for the shell-buffer, the *prompt-end-character*, and shell type.

Create Default System Shell

If the Access facility is loaded and the System Shell does not yet exist, this command will create it using the shell type specified in the Options browser. This command is also available through the Options Browser and popup menus.

Create System Shell

This command differs from the **Create Default System Shell** command only in that it prompts you for the shell type.

Create Default Shell Buffer

If you currently have less than 10 shells, this command will create a new shell-buffer using the Options browser values. Your location is left unchanged.

This command is also available through the Options browser and popup menus.

Create Shell-Buffer
This command is the same as **Create Default Shell-Buffer, except that you are prompted for the buffer name,** *prompt-end-character*, and shell type.

Kill All HP-UX Shells
Browsing this function prompts to check if you really want to kill all of the shells. You may answer yes or no. Attempting to use an HP-UX/NMODE command without having a shell will produce an error.

This command is also available through the Options browser and popup menus.

Kill System Shell
Executing **M-X Kill System Shell** kills the System Shell. (You can create a new system shell if desired.) Any shell-buffers you have created will still work correctly, but you will not be able to communicate with HP-UX by using the NMODE/HP-UX commands from other NMODE buffers.

This command is also available through the Options browser and popup menus.

Kill Shell-buffer Shell
Browsing this option prompts you for the name of the shell-buffer to kill. Before you browse this item, you may wish to visit the buffer browser (**C-X C-B**) to see the names of the shell-buffers. Each shell-buffer will have an `S` indicator to the left of its name. If you have the System Shell, you can still use NMODE/HP-UX commands from other buffers.

This command is also available through the Options browser and popup menus.

Kill Current Shell-Buffer
If you are currently located in a shell-buffer, this command will kill it and leave you at your previous location.

This command is also available through the Options browser and popup menus.

The HP-UX Access Facility Model

Here is the model of the HP-UX Access Facility. The following two sections show the two ways shells are used in the facility.

The Shell-Buffer

The following illustration shows the shell-buffer interface. It behaves similar to a regular Bourne shell or *csh* but since it is an NMODE text buffer, all text typed and all HP-UX responses are saved in the buffer.

```
 ------------
|            |
|  Buffer    |   HP-UX
|            |   Mode
 ------------
     |  ^
     v  |
 ------------         ------------         ------------         -----------
|            |       |            |       |            |       |           |
|   NMODE    | ->    |  Master    | ->    |   Slave    | ->    |  HP-UX    |
|   Shell    |   <-  |   Pty      |   <-  |   Pty      |   <-  |  Shell    |
|            |       |            |       |            |       |           |
 ------------         ------------         ------------         -----------
                                                                   |  ^
                                                                   v  |
                                                                ------------
                                                                   HP-UX
```

Shell-Buffer Interface

When you type something and press [Return], NMODE takes what you typed and passes it (through an HP-UX pty) to a shell. The response from HP-UX is then passed back (through the pty) and appended to the buffer.

Note that the master and slave pty provides the communication path between the NMODE shell and an HP-UX shell. See *pty(4)* in the *HP-UX Reference* if you need more information about how they work.

Also note that the buffer has an HP-UX minor mode that provides commands that supplement those in Emacs mode and Text mode.

The System Shell

Sometimes it is useful to be able to send a command to HP-UX from any buffer, not just one that is a shell-buffer. The HP-UX Access Facility provides a System Shell to which you can direct any piece of text from any buffer.

The System Shell is shared by all buffers. This implies that if one buffer is using the System Shell, a command from another will be queued (by the the shell).

```
    ------------        ------------        ------------        -----------
   |            |      |            |      |            |      |           |
   |   Buffer   |      |   Buffer   |      |   Buffer   |      |   Buffer  |
   |            |      |            |      |            |      |           |
   |            |      |            |      |            |      |           |
    ------------        ------------        ------------        -----------
        |                    |                   |                   |
        |                   /                   /                   /
        |                  /                   /                   /
        -------<-------------------<--------------------<--------
        |
        v
    ------------        ------------        ------------        -----------
   |            |      |            |      |            |      |           |
   |   NMODE    | ->   |   Master   | ->   |   Slave    | ->   |   HP-UX   |
   |   Shell    |   <- |    Pty     |   <- |    Pty     |   <- |   Shell   |
   |            |      |            |      |            |      |           |
    ------------        ------------        ------------        -----------
       |                                                           |  ^
       x HP-UX Set Output Buffer command                           v  |
       |                                                        -----------
       v                                                           HP-UX
    ------------
   | HP-UX      |
   | Output     |
   | Buffer     |
   |            |
    ------------
```

System-Shell Interface

In using the system shell, NMODE accepts input from a buffer and appends output to the HP-UX Output buffer. Again, the master and slave pty provides the communication path between the NMODE shell and an HP-UX shell.

Possible Problems and Anomalies

The purpose of the HP-UX Access facility is to provide a convenient way for you to execute most HP-UX commands without leaving the NMODE user environment. As you may know, there are many types of HP-UX commands and some of them insist upon being "connected" to a terminal. Such commands are not "fooled" by the techniques used in the access facility. The facility works like a dumb terminal (see *termcap(3X)* or *terminfo(5) in the HP-UX Reference* if you want additional information about general characteristics).

Rules to Remember

Here are a few things that you should remember when you use the HP-UX Access facility.

- Be sure that all commands are typed on one line.
- Do not try to "type-ahead" while results are being output.
- Do not use Control, Escape, or other special characters expecting them to be processed before the shell sees them.
- Do not change the System Shell's environment.
- Avoid setting marks when using a shell-buffer In general, the system expects a mark at the beginning of a command. If your mark seems to be confused, and you are not using the prompt-end-characteı option in a shell-buffer, simply type **C-C**.
- The facility uses a subset of *csh*. HP extensions to the standard *csh* are not available.
- The PTYs (HP-UX device files) used by the facility are shared by Windows/9000. This can result in the window-manager not being able to create another window. The conflict can be fixed by directing the window-manager to use a higher numbered set of PTYs since it insists that the PTYs be in a contiguous block.
- If HP-UX is heavily loaded, extraneous prompts may appear in the output. This is due to NMODE and the shell being "out of sync" with each other.
- The HP-UX Filter Region command used with the System Shell, changes the setting of the designated output buffer. To change again, use the HP-UX Set Output Buffer command.
- The HP-UX Filter Region command used in a shell-buffer will print your prompt as part of its output since NMODE does not post-process shell output.
- When using any HP-UX Region command in a shell buffer, the prompts will also be sent with the rest of the text if you include the prompt in the region.
- Do not press [Break] during the creation of a shell. It will render the shell useless and reduce the number of available shells.

Unresponsive Shell-Buffers

If there is no response to a command, it might be that an incomplete command or an improper command was sent to the shell. After waiting a reasonable amount of time, you can issue the **HP-UX Send Interrupt** command to interrupt the shell (**C-C** in a shell-buffer, **M-H C-C** in any other buffer).

Accidental Logout

If you accidentally logout of the shell-buffer or if it appears to quit working, you can try one or both of the following recovery procedures.

Execute the **M-X Recover Shell-Buffer Shell** command. You will be asked for the name of the "dead" shell-buffer, the unique prompt character, and the "type" of shell (*sh* or *csh*). The system will then try to "reconnect" the shell-buffer to a new HP-UX shell.

Of course, it is also possible to kill and recreate any or all of the shells used by the facility. If you wish to recreate shells, you can follow these steps.

1. Go to NMODE Root. (**C-X R**)
2. Browse `User Options`.
3. Browse `HP-UX Access Options`.
4. Browse `Kill Shell-Buffer`. You will be asked for the name of the shell to kill. If you do not know the name of the shell, you can either check the list of buffers (use `C-X C-B`) and determine the correct name, or you can simply browse the `Kill All HP-UX Shells` option.
5. Browse the `Create new shell-buffer only` option to create a new shell. If you killed all of the shells in the previous step, simply browse the `Create System Shell and shell-buffern` option instead.

If both of the recovery procedures fail, you may wish to reboot NMODE and reload the facility.

Chapter 13
Code Indexes

Introduction

The code index facility offers the user a means of viewing their source code from a higher level. A code index allows the user to obtain a skeletal view of their source code. The code index of a source file acts like an index into that file. You can have code indexes for Lisp, C, Pascal and Fortran source code.

Typically, the items of a code index are routine headers, each item containing the name of each routine available within the designated source files. However, you have the option to choose which language components, or code forms, you deem necessary for skeletal viewing.

This facility serves as an index into code as follows. You create a code index that will display the top level code forms of a designated file or set of files. Items are placed into the code index by reading the files and extracting the relevant lines of code. All items of a file are listed under that file name, where the file name also serves as an item in the code index. Once you find an item of interest, selecting the item and browsing in will place you in a buffer containing the source file at the location of that item in the file, ready for viewing or editing.

As a software project writes numerous routines and needs to make use of existing ones, tracking these can be made easier by the availability of code indexes of the source files. The task of finding a particular routine is reduced to searching through a list, with browsing commands (such as `Filter`) available to operate on this list.

There is also a Find Item feature that allows the user to point to a word in a buffer and request the system to enter the file containing the code form that matches the word pointed at the location of the code form. Typically this is used to move from a call of a routine to the definition of that routine.

Loading the Browser for Code Indexes

To work with and execute code index commands, you need to load the code that implements them. This can be done in two different ways. The simplest way is to browse into Additional Facilities, put the cursor on the `Code Index` item, and execute either of the `Load` or `Browse/execute` command line commands. The difference between the two is that `Browse/execute` assumes that you want to create a new code index immediately after loading.

You can also load the code index support functions by evaluating a Lisp form. If you expect to use this type of browser often, you can have the code index facility loaded automatically by adding this form to your make-nmode or initialization file (`.nmoderc` in your home directory):

```
(require "code-browser")
```

The form sets up the facility in `NMODE Root` called `Code Indexes`, which lets you access the active code indexes. While this loads the facility, you will not necessarily see any code indexes displayed on the screen when you are in the facility. You need to complete a few more steps.

Creating New Code Indexes

After having loaded the code index facility, you can now create a code index. One method is to create an empty code index first and then explicitly add files to it. To do this,

- You can browse Additional Facilities and use the `Browse/execute` command.
- You can execute **M-X create code index** from anywhere in the system.
- From the Code Index Facility or within a code index itself, you can use `Create` and its sub-command `Code-index`.

These will prompt for the name of a new code index. After you enter the name, the system will create and then enter the new code index.

At this point, you need to add items to the code index and then save it if you want to use the code index beyond the duration of your current session. To add new items to the code index, you should execute the command named `Add-files` (as described in a later section).

Another method of creation consists of creating a code index and building the items via one command. This can be done by

- Executing **M-X browse code** from within a buffer.
- Using `Type-specific` in a directory browser with sub-command `Browse-code` on a file with suffix `.l`, `.p`, `.c`, or `.f` (for source code in Lisp, C, Pascal or Fortran). In this case the code index will have the same name as the file whose code was browsed.

This last method also has a means of handling multiple files at one time. This can be done by grouping the files you desire within the directory browser. Then use the `Type-specific` `Browse-code` with the current item being one of the files in the group. This will prompt for the name of the code index and then create the items from each grouped file.

Code Index Facility

At the top level of NMODE, you can point to and browse into the facility called `Code Indexes`. This invokes a browser that will contain items for each of the code indexes in the NMODE system. Like working with the browser for active directories, notice the implicit two-step procedure: the facility provides a browser that lets you browse into a particular code index.

The screen display might look like this:

```
Code Indexes

    acme-code



 @ Browser (Code Indexes)
Help  Browse  Group  Filter  Create  Options  Kill  Quit
```

Note that the title and mode line indicate your location in NMODE and you have a menu of commands. The important thing is the item in the browser, `acme-code` in this case. If you select an item in this browser and invoke the `Browse` command you will enter the corresponding code index. Each item in a code index is a pointer to where that code form appears in a source code file.

A Code Index

The previous example of the Code Indexes Facility contained an item called `acme-code`. Pointing to and browsing into the `acme-code` code index takes you into that particular set of items. Each item in this database lets you access the source code file that contains the code form.

The screen display might look like this:

```
     Code Index: acme-code

         /users/guest/progs/demo.l

         ;; Demo for code index, etc
         (defun foo (a b)



      @ Browser (Code Index) acme-code

Help  Browse  Group  Create  Utilities  Options  Add-files  Quit
```

Like all browsers, you see a title and mode line that indicate your location in NMODE and you have a menu of commands that are discussed later. There are actually three items in this code index. That is:

```
     /users/guest/progs/demo.l

     ;; Demo for code index, etc
     (defun foo (a b)
```

are the three index items that point into the source file named `demo.l`. You can point to and browse any of the three lines (the file name or either code form item) and NMODE will find the source code file, load it into memory, and visit the file in a buffer according to the following scheme:

- If you point to and browse `/users/guest/`..., NMODE locates you at the beginning of the file.
- If you point to and browse the comment, `;; Demo` ..., NMODE locates you at the comment in the associated source file.
- If you point to and browse `(defun foo` ..., NMODE locates you at the beginning of the defun in the associated source file.

Regardless of your location within the buffer, you can edit the file as desired. Note that the above display focused on an extremely simple situation to illustrate the idea. Actually, code indexes will normally contain many items for a single file and might contain several files. In general, the list for a code index looks like this:

```
 file-name-1
```

item-1
item-2
.
.
.
item-n

file-name-2

The list of items can become very long. Use the assorted commands, such as `Filter`, in the menu to manipulate the list of items so you can obtain manageable sublists. The sections on commands discuss this.

The displayed items in a code index that you develop yourself could vary from what was shown. The actual items that get displayed depend on the values assigned to the user options for the browser of code indexes. If you want to change how code forms are extracted from source files, you can execute the command named `Options` and set the values for options according to your needs. This is discussed later in the section called "The Options Command (Code Indexes)".

Find Item Command

A very useful command is the `Find Item` command, which can be invoked via **M-.** or **M-X find item.** With the cursor on a word in the current buffer, invoking **M-.** will cause a search for this word throughout all the code indexes in the system. If there is an item that matches the word, then the source file will be displayed as if you had browsed into the item yourself from a code index. Thus you will be placed at the location of the relevant code.

When you are in a buffer viewing some source code, and there is a function being called for which you want to see the definition, put the cursor on the function name in the buffer and invoke **M-.**. NMODE will prompt for the name to search for. Notice that the default value will be the word the cursor was positioned on in the buffer. Press Return in response to the prompt. This initiates a search for the source code that defines the function. It does this by searching for the function name through all code indexes in the system. When it finds this function name in an item, it will browse into the item for you, displaying the function definition.

If for some reason there are multiple items among the code indexes with this name, you will be presented with a miniature code index of the matching items listed under their source file names. You could then browse into an item to see the code associated with that item.

Using the example of demo.l from the section before, the function being called in code you are viewing in a buffer might be `foo`. Putting the cursor on any letter of this word in your buffer and issuing **M-.** will cause the prompt:

```
    Type in the name of the Item you want to see. (Default is: 'foo')
```

to appear. Pressing Return initiates a search for the function definition of `foo`. It will find it in code index acme-code and place you in a buffer of the file /users/guest/progs/demo.l, displaying at the top "(defun foo (a b) ...".

If the cursor is not positioned at a word, the default will be the default value used in the previous invocation of the `Find Item` command. You can override the default it picks up by entering a string for a search of a different item.

As you use `Find Item`, you may travel down several levels. Maybe you are at a definition of a function via **M-.** and you wish to view the definition of another function being called by this current one. You can invoke another **M-.** and view that definition. At any time you can exit back through the levels to the original buffer you started at by invoking **C-M-L** at each level you have visited. You will then be positioned at the first function call you invoked `Find Item` on.

Another example might be in Pascal source, where a global variable is being used and knowing its type would be useful. Having loaded in source code with the option of `Variable Extraction` turned on (options will be discussed in the next section), put the cursor on this variable and issue the M-. command. If it is defined in the set of source code the code indexes in your NMODE consist of, its definition will be displayed. Type definitions can similarly be displayed, having chosen the `Type Extraction` option in the Pascal Options Browser.

Also note that you can change a buffer or file after creating the code index, such that lines that serve as items in the code index are now in a different location in the buffer or file. `Browse` and `Find Item` commands will still work as expected. In most cases, you will not have to update the code index. However, if any of these lines are themselves changed, the following could happen:

- `Find Item` is not able to find the item and the system informs you that it is unable to find the source code for it.
- `Find Item` finds the item, but is unable to find the line of the item in the source file and the system informs you that the code index refers to an item no longer in the file.
- You attempt to `Browse` into an item whose line of code no longer exists in the file and the system beeps, remaining in the code index.

When any of these occur, you will need to update the code to reflect the changed file. Updating will be described under the section "Executing Browser Commands (Code Indexes)"

Executing Browser Commands (Code Indexes)

A code index browser has the following menu of commands:

```
    Help  Browse  Group  Filter  Create  Utility  Options  Add-files  Quit
```

The commands named `Help`, `Browse`, `Group`, `Filter`, `Create`, and `Quit` were described earlier in the section called "Common Browser Commands" in the chapter called "Introduction to Browsers". The next few sections discuss the other commands.

The Add-files Command (Code Index)

The `Add-files` command allows you to add to the list of files that are being indexed in a particular code index. During execution, the command processes a source code file and extracts code forms such as the initial lines in `defuns` or `define-methods`. Those items are references to the complete code forms in the file, and appear in the code index's list under the file name from which they were extracted. Use the `Add-files` command when you make a code index or when you want to add more files to an existing index. Add those files that have an appropriate suffix: `.l`, `.p`, `.c`, or `.f`.

The `Add-files` command prompts:

```
    File (or "wildcard" spec): (relative to '/users/guest/')
```

Type the file name, or path name for source code files, that you want to add to the code index. As examples:

```
    acme/code22.l
```

adds indexes from one Lisp source file, `code22.l`, from the directory named `acme` under `/users/guest/`. Using the wildcard character "*" allows multiple files to be specified. Entering

```
    /users/joint-project/ace/*.c
```

adds indexes for all the C Source files in the directory named ace under `/users/joint-project/`.

If you specify a directory name only, then all of the files in the directory will be added.

You can add indexes for as many files as you want up to the capacity of your system.

The Utilities Command (Code Index)

Executing `Utilities` provides the following menu of second-level commands:

```
    Update   Kill   Write   Current-File   Quit-utilities
```

These sub-commands work as follows:

`Update`	Reads the current file associated with the current item in the code index and updates the items in the code index to reflect the actual state of the source files. If you want more than one file updated, group the file name items of the desired files in the code index. Then select one of the grouped items and invoke `Utilities Update`.
`Kill`	Removes the highlighted item from the code index. This only affects the code index and has no effect on the actual source code file. If the highlighted item is a file name, all items of that file and the file name will be removed.
`Write`	Saves the code index into a file. Then it can be loaded (or registered) in a subsequent NMODE session. See the section "Saving and Restoring Code Indexes" later in this chapter for further details.
`Current-file`	Many source code files contain more interesting items than can be displayed in a single pane. Executing `Current-file` displays in the message area the name of the source file for the current highlighted item.
`Quit-utilities`	Exits the Utilities command and returns to the code index.

The Options Command (Code Index)

There is a chapter devoted to user options called "The Browser for User Options". That chapter discusses how to access and use options browsers in general. You may want to refer to that chapter as this section will describe options pertaining to code indexes.

Executing `Options` provides the following menu of second-level commands:

```
    Lisp   Pascal   C   Fortran   Quit-options
```

Choose the type of Code Index you wish to customize. This will place you in an options browser for that language. This browser provides the following menu of commands.

```
Help Browse/modify Group Filter Sort Write Restore-default Quit
```

The commands named `Help`, `Group`, `Filter`, `Sort`, `Write`, and `Quit` are familiar from previous chapters dealing with browsing in general. Executing `Browse/modify` lets you alter the value of the current option. Executing `Restore-default` assigns the default value to the current option.

The following shows the options (system variables) for each language. Each list depicts what you see on the screen when you invoke the related user options browser. In each case, you see the assigned default values for options.

If you execute `Options` and then execute `Lisp`, you see:

```
User Options: LISP Code Index Options
  Currently saved in "$LISP/config/lisp-code.opt"

CODE INDEX OPTIONS:
     Function Extraction           Yes
     Comment Header Displayed      Yes
     Form Type to Extract          NIL
```

`Function Extraction` causes any line with a left parenthesis, "(", in column 0 to be be extracted. For the code index to process functions correctly, the left parenthesis of the function definition must appear in column 0.

`Comment Header Displayed` will display comment lines that are each preceded by a line of at least five comment characters (e.g. ";;;;;"). So a code index for

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Printing Aids:
; Other comments may follow

(defun print-caps ()
  ...
```

will contain the line "; Printing Aids:" as an item.

`Form Type to Extract`. The value of this option is a string type which names a form to be displayed in the code index. The entire form (whose left parenthesis is in column 0), not just one line, will be extracted from the source. For example, a value of "export" will cause each occurrence of (export ...) to be displayed in its entirety. So

(export '(terminal-default single-window window-width window-height)) would appear in the code index.

If you execute Options and then execute Pascal, you see:

```
User Options: Pascal Code Index Options
  Currently saved in "$LISP/config/pscl-code.opt"

CODE INDEX OPTIONS:
     Procedures and Functions          Yes
     Constant Extraction               No
     External Routine Declarations     No
     Import Extraction                 No
     Include Extraction                No
     Module Extraction                 No
     Type Extraction                   No
     Variable Extraction               No
```

The default is procedure and function headers becoming items in the code index. Choosing the other options with a Yes value will cause those lines of code to be displayed in the code index. For example, choosing Type extraction will cause the whole global type section to be displayed in the code index, with each type definition becoming an item. Similarly, Constant Extraction or Variable Extraction cause entire CONST or VAR sections to be displayed.

Choosing External Routine Extraction will display the procedure and function headers designated "external" in the source code.

Choosing Import Extraction will cause the appearance of IMPORT statements. Choosing Module Extraction will cause the lines of MODULE, EXPORT and IMPLEMENT to become items in the code index. And choosing Include Extraction will cause include lines to become part of the code index (note the Browse Include **C-M-B** command described below).

If you execute Options and then execute C, you see:

```
User Options: C Code Index Options
  Currently saved in "$LISP/config/c-code.opt"

BLOCK DELIMITER OPTIONS:
     Block begin character(s)        "{"
     Block end character(s)          "}"

EXTRACTION OPTIONS:
     Define lines                    No
     Function extraction             Yes
     If directives                   No
     Include directives              No
```

```
        Typedef extraction            No
        Variable extraction           No
```

Function extraction is the default. The first line of each function declaration will become an item in the code index. If other options are chosen with value of `Yes`, those source code items will also be extracted from each file and displayed in the code index. For example, choosing Variable extraction will cause the global VAR section to be placed in the code index. Choosing `Define lines` or `Include directives` will cause the #define lines or #include lines to appear. (Note the Browse Include **C-M-B** command described below). A value of `Yes` for `If directives` will extract those lines with #if, #ifdef, #ifndef, #else or #endif. Choosing `Typedef extraction` will cause entire global typedef statements to appear in the code index.

Notice also that since the C language allows redefinition of the blocking characters "{" and "}", the option browser fives the user a chance to specify how they were changed. This allows the code index to still find function declarations with new values. For example, "Begin" and "End" could be used in place of "{" and "}".

If you execute `Options` and then execute `Fortran`, you see:

```
User Options: Fortran Code Index Options
  Currently saved in "$LISP/config/ftn-code.opt"

EXTRACTION OPTIONS:
     Common Declarations              No
     Data Statements                  No
     Dimension Statements            No
     Functions and Subroutines        Yes
     Include Statement/Directives     No
     Parameter Lists                  No
     Types Declarations               No
```

The default for Fortran code indexing is the subroutine and function lines of each file becoming items in the code index. Other lines of code can also become items of the code index by changing the value to `Yes`. For example, by choosing `Types extracted`, source code lines declaring variables to be of the types INTEGER, REAL, DOUBLE PRECISION, DOUBLE COMPLEX, CHARACTER or COMPLEX LOGICAL will be displayed. `Common Extraction`, `Data Extraction`, `Dimension Extraction`, `Include Statement/Directives`, or `Parameter Extraction` cause COMMON lines, DATA lines, DIMENSION lines, INCLUDE lines, or PARAMETER lines respectively, to become items in the code index.

Browse Include Command

With include statement items for any language, you can browse directly into the actual include file. First select the include statement item in the code index. Now invoke **C-M-B**. NMODE will not enter the source file that contains the include statement. Instead it will enter the include file itself. If **C-M-B** is invoked when the current item is not an include statement, the source file of the item will be browsed as normal.

Saving and Restoring Code Indexes

When an NMODE session is exited all code indexes that were created during that session are destroyed. To save the effort of having to create code indexes each time a new NMODE session is started there are commands that allow you to save a code index after it has been created and restore that code index in a subsequent session.

Saving a Code Index

To save a code index you must be in the code index you want to save. Execute `Utilities` and `Write` to save the code index. NMODE will display a prompt something like:

```
    Write browser database to file (Default is '$HOME/acme-code.cb')
```

If you enter a Return in response to the prompt the default value is used as the file name to save to. If you enter the path name of a directory followed by "/" then the code index is saved in the file in that directory. In the example above if you entered **/users/guest/code-indexes/** the code index would be saved in the file **/users/guest/code-indexes/acme-code.cb**. If you enter the path name to a file NMODE will modify the name of the file to insure that it is not longer than 14 characters and that it ends in ".cb". The code index be will saved in the file with the resulting name. Saved code index file names will always end in the suffix `.cb` so they can be identified.

After a response is entered NMODE will display the message:

```
    Writing index database ...
```

When the save operation is complete NMODE will display a message of the form:

    File *file-name* written.

to indicate that the save operation is complete.

Loading Saved Code Indexes

A code index that has been created and saved can be loaded back into NMODE either interactively or programmatically. There are several methods that can be used to load a saved code index interactively.

- Within the `Code Indexes` facility you can invoke the `Create` command. NMODE will display a new menu in the prompt area. If you now invoke the command `Load-code-index` NMODE will display a prompt similar to:

  ```
  Load index file (type defaults to "cb"): (relative to /users/guest/)
  ```

  You should enter the name of the file containing a saved code index. You do not need to enter the full path name of the file. The suffix can be omitted. It will default to `.cb` as the prompt `(type defaults to "cb")` indicates. Also the path name can be entered relative to the working directory of NMODE, which is displayed after `relative to` in the prompt. After you enter a file name NMODE will load the code index saved in the file into the system and move you into that code index. If the file does not exist or is not a saved code index NMODE will display the error message:

  `File` *file-name* `is not a saved code index`

- You can invoke the command **M-X Load Code Index** to load a saved code index from anywhere within NMODE. NMODE will respond with the same prompt discussed above. After you enter the file name of a saved code index NMODE will load and enter the code index.

- When you are in a directory browser containing items for files that are saved code indexes these code indexes can be loaded using directory browser commands. First you must select an item for a file containing a saved code index that you want to be loaded. Now you can load the code index by invoking either the `Browse` or the `Type-specific Load` command. NMODE will then load the code index from the indicated file and then enter the code index. These two commands can be used on grouped items to load more than one code index. If the current item is part of a group then invoking the `Browse` or `Type-specific Load` commands will load saved code indexes for each of the files in the group.

You may want to load saved code indexes during the initialization of your NMODE session. For this reason there is a programmatic method of loading saved code indexes. If you evaluate the form:

`(nmode:load-file-index-browser-db` *file-name*`)`

NMODE will load a saved file index from *file-name*. This function is used to load saved code, file search and error indexes programmatically. File index is a generic term that covers all three of these indexes. *File-Name* must be the full path name to the code index file and must end in `.cb` when loading a saved code index. If the file does not exist or is not a saved code index NMODE will display the error message discussed above.

The function `nmode:load-file-index-browser-db` can be invoked from your NMODE initialization or customization files to load code indexes when your NMODE session is started.

When saved code indexes are loaded programmatically the code indexes are not automatically entered. However, they will be displayed in the `Code Indexes` facility just like other code indexes.

Registering Saved Code Indexes

Often you will only want to use a code index with the **M-.** command. You can use this command without loading the code index into memory. This can be done by registering a saved code index. When a code index is registered, an entry for it is added to the list of code indexes that appear in the `Code Indexes` facility. Invoking **M-.** will search through all code indexes appearing in the facility. The advantage of registering a code index rather than loading it is that a registered code index does not use as much memory as a loaded code index.

You can register a saved code index in several ways that are similar to the ways in which a code index can be loaded.

- Within the `Code Indexes` facility you can invoke the `Create` command. If you now invoke the command `Register-code-index` NMODE will display a prompt similar to:

  ```
  Register index file (type defaults to "cb"): (relative to $HOME/)
  ```

  You should enter the name of the file containing a saved code index. You can omit the suffix and supply a relative path name as with the corresponding command for loading a code index. After you enter a file name NMODE will register the code index saved in the file. If the file does not exist or is not a saved code index NMODE will display the error message:

  `File` *file-name* `is not a saved code index`

- You can invoke the command **M-X Register Code Index** to register a saved code index from anywhere within NMODE. NMODE will respond with the same prompt discussed above. After you enter the file name of a saved code index NMODE will register the specified code index.

- When you are in a directory browser containing items for files that contain saved code indexes, these code indexes can be registered using a directory browser command. First you must select an item that is a file containing a saved code index that you want to be registered. Now you can register the code index by invoking the `Type-specific Register` command. NMODE will then register the code index from the indicated file. This command can be used on grouped items to register more than one code index by invoking one command. If the current item is part of a group then invoking the `Type-specific Register` command will register saved code indexes for each of the files in the group.

A registered code index can be loaded in a way not discussed in the section on loading code indexes. If you are in the `Code Indexes` facility you can select an item corresponding to a registered code index. Then if you invoke the `Browse` command the code index will be loaded and will no longer be registered.

Chapter 14
Search Indexes

Introduction

Search indexes are similar to code indexes in that they extract lines from files to make items in a browser. What makes a search index different is that it extracts all the lines from files that contain a string pattern being searched for. Selecting an item in a search index and browsing in will place you in a buffer containing the file that the line of the item was extracted from. The cursor will be placed at the beginning of the line for the item that was browsed. This provides a capability similar to the HP-UX *grep(1)* command with the additional feature of interacting with the index after it is built.

This facility serves as an index into text as follows. You create a search index that will display the lines containing a given string pattern, from a designated file, or set of files. Items are placed into the search index by reading the files and extracting the relevant lines of text. All items of a file are listed under that file name, where the file name also serves as an item in the search index. Once you find an item of interest, selecting the item and browsing in will place you in a buffer containing the text file at the location of that item in the file, ready for viewing or editing.

Loading the Browser for Search Indexes

To work with and execute search index commands, you need to load the code that implements them. This can be done in two different ways. The simplest way is to browse into Additional Facilities, put the cursor on the `Search Index` item, and execute either of the `Load` or `Browse/execute` command line commands. The difference between the two is that `Browse/execute` assumes that you want to create a new search index immediately after loading.

You can also load the search index support functions by evaluating a Lisp form. If you expect to use this type of browser often, you can have the search index facility loaded automatically by adding this form to your make-nmode or initialization file (`.nmoderc` in your home directory):

```
(require "search-br")
```

The form sets up the facility in `NMODE Root` called `Search Indexes`, which lets you access the active search indexes. While this loads the facility, you will not necessarily see any search indexes displayed on the screen when you are in the facility. You need to complete a few more steps.

Creating New Search Indexes

After having loaded the search index facility, you can now create a search index. One method is to create an empty search index first and then explicitly add files to it. To do this,

- You can browse Additional Facilities and use the `Browse/execute` command.
- You can execute **M-X create search index** from anywhere in NMODE.
- From the Search Index Facility or within a search index itself, you can use `Create` and its sub-command `Search-index`.

These will prompt for the name of a new search index. After you enter the name, NMODE will create and then enter the new search index.

At this point, you need to add items to the search index and then save it if you want to use the search index beyond the duration of your current session. To add new items to the search index, you should execute the command named `Add-files` (as described in a later section).

Another method of creation consists of creating a search index and building the items via one command. This can be done by

- Executing **M-X browse occurrences** from within a buffer containing a file. In this case the search index will have the same name as the file in the buffer.

Search Index Facility

At the top level of NMODE, you can point to and browse into the facility called `Search Indexes`. This invokes a browser that will contain items for each of the search indexes in the NMODE system. Like working with the browser for active directories, notice the implicit two-step procedure: the facility provides a browser that lets you browse into a particular search index.

The screen display might look like this:

```
Search Indexes

    tab-search



 @ Browser (Search Indexes)
Help  Browse  Group  Filter  Create  Options  Kill  Quit
```

Note that the title and mode line indicate your location in NMODE and you have a menu of commands. The important thing is the item in the browser, `tab-search` in this case. If you select an item in this browser and invoke the `Browse` command you will enter the corresponding search index. Each item in a search index is a pointer to where that string pattern appears in a text file.

A Search Index

The previous example of the Search Indexes Facility contained an item called tab-search. Pointing to and browsing into the tab-search search index takes you into that search index. Each item in the search index lets you access the text file that contains the search pattern.

With the search pattern set to **tab**, the screen display might look like this:

```
Search Index: tab-search

    /users/guest/progs/document.text

    with the tab set to 0 you can now
    the tab is set to a number greater than




 @ Browser (Search Index) tab-search

Help  Browse  Group  Create  Utilities  Options  Add-files Pattern Quit
```

Like all browsers, you see a title and mode line that indicate your location in NMODE and you have a menu of commands that are discussed later. There are actually three items in this search index. That is:

```
/users/guest/progs/document.text

with the tab set to 0 you can now
the tab is set to a number greater than
```

are the three index items that point into the text file named document.text. You can point to and browse any of the three lines (the file name or either text item) and NMODE will find the text file, load it into memory, and visit the file in a buffer according to the following scheme:

- If you point to and browse /users/guest/..., NMODE locates you at the beginning of the file.
- If you point to and browse with the tab..., NMODE locates you at that line in the associated text file.
- If you point to and browse the tab is..., NMODE locates you at the that line in the associated text file.

Regardless of your location within the buffer, you can edit the file as desired. Note that the above display focused on an extremely simple situation to illustrate the idea. Actually, search indexes will normally contain many items for a single file and might contain several files. In general, the list for a search index looks like this:

```
 file-name-1
```

item-1
item-2
.
.
.
item-n

file-name-2

The list of items can become very long. Use commands, such as `Filter`, in the menu to manipulate the list of items so you can obtain manageable sublists.

Executing Browser Commands (Search Indexes)

A search index browser has the following menu of commands:

```
 Help  Browse  Group  Filter  Create  Utility  Options  Add-files Pattern Quit
```

The commands named `Help`, `Browse`, `Group`, `Filter`, `Create`, and `Quit` were described earlier in the section called "Common Browser Commands" in the chapter called "Introduction to Browsers". The next few sections discuss the other commands.

The Pattern Command

The `Pattern` command allows you to enter the string pattern to be searched for. When this command is invoked NMODE will issue the prompt

```
    New search pattern
```

You should now enter the exact characters of the pattern you want to search for and then enter Return. All the files that have been added to the search index are searched for the pattern and the lines containing the pattern will be added the search index under the corresponding file name.

After a pattern has been entered if you invoke the pattern again the prompt displayed will be some what different. It will have the form:

`New search pattern (Default is: '`*current search pattern*`')`

Notice that the current search pattern is displayed as the default value. If you just press Return the search pattern will remain the default. Using the `Pattern` command in this way allows you to determine what search pattern is associated with a particular search index. You can also change the search pattern and NMODE will search all the files in the search index as discussed above.

The Add-files Command (Search Index)

The `Add-files` command allows you to add to the list of files that are being indexed in a particular search index. During execution, the command processes a text file and extracts the lines containing the search pattern. The resulting index items are references to the lines containing the search pattern in the file, and appear in the search index's list under the file name from which they were extracted. Use the `Add-files` command when you make a search index or when you want to add more files to an existing index.

The `Add-files` command prompts:

```
    File (or "wildcard" spec): (relative to '/users/guest/')
```

Type the file name, or path name for text files, that you want to add to the search index. As examples:

```
    manual/tab.text
```

adds indexes from one text file, `tab.text`, from the directory named `manual` under `/users/guest/`. Using the wildcard character * allows multiple files to be specified. Entering

```
    /users/joint-project/ace/*.text
```

adds indexes for all the text files in the directory named `ace` under `/users/joint-project/`.

If you specify a directory name only, then all of the files in the directory will be added.

The Utilities Command (Search Index)

Executing `Utilities` provides the following menu of second-level commands:

```
    Update  Kill  Write  Current-File  Quit-utilities
```

These sub-commands work as follows:

`Update`	Reads the current file associated with the current item in the search index and updates the items in the search index to reflect the actual state of the search pattern in the text files. If you want more than one file updated, group the file name items of the desired files in the search index. Then select one of the grouped items and invoke `Utilities Update`.
`Kill`	Removes the highlighted item from the search index. This only affects the search index and has no effect on the actual text file. If the highlighted item is a file name, all items of that file and the file name will be removed.
`Write`	Saves the search index into a file. Then it can be loaded (or registered) in a subsequent NMODE session. See the section "Saving and Restoring Search Indexes" later in this chapter for further details.
`Current-file`	Many text files contain more interesting items than can be displayed in a single pane. Executing `Current-file` displays in the message area the name of the text file for the current highlighted item.
`Quit-utilities`	Exits the Utilities command and returns to the search index.

The Options Command (Search Index)

This command loads in the NMODE General User Options. There is an chapter devoted to user options called "The Browser for User Options". That chapter will discuss how to access and use options browsers in general. You may want to refer to that chapter.

Saving and Restoring Search Indexes

When an NMODE session is exited all search indexes that were created during that session are destroyed. To save the effort of having to create search indexes each time a new NMODE session is started there are commands that allow you to save a search index after it has been created and restore that search index in a subsequent session. Both the items in a search index and the search pattern associated with it are saved and restored.

Saving a Search Index

To save a search index you must be in the search index you want to save. Execute `Utilities` and `Write` to save the search index. NMODE will display a prompt something like:

```
Write browser database to file (Default is '$HOME/tab-search.sb')
```

If you enter a Return in response to the prompt the default value is used as the file name to save to. If you enter the path name of a directory followed by / then the search index is saved in the file in that directory. In the example above if you entered **/users/guest/search-indexes/** the search index would be saved in the file **/users/guest/search-indexes/tab-search.sb**. If you enter the path name to a file, NMODE will modify the name of the file to insure that it is not longer than 14 characters and that it ends in `.sb`. The search index be will saved in the file with the resulting name. Saved search index file names will always end in the suffix `.sb` so they can be identified.

After a response is entered NMODE will display the message:

```
Writing index database ...
```

When the save operation is complete NMODE will display a message of the form:

File *file-name* written.

to indicate that the save operation is complete.

Loading Saved Search Indexes

A search index that has been created and saved can be loaded back into NMODE either interactively or programmatically. There are several methods that can be used to load a saved search index interactively.

- Within the `Search Indexes` facility you can invoke the `Create` command. NMODE will display a new menu in the prompt area. If you now invoke the command `Load-search-index` NMODE will display a prompt similar to:

  ```
  Load index file (type defaults to "sb"): (relative to /users/guest/)
  ```

  You should enter the name of the file containing a saved search index. You do not need to enter the full path name of the file. The suffix can be omitted. It will default to `.sb` as the prompt `(type defaults to "sb")` indicates. Also the path name can be entered relative to the working directory of NMODE, which is displayed after `relative to` in the prompt. After you enter a file name NMODE will load the search index saved in the file into NMODE and move you into that search index. If the file does not exist or is not a saved search index NMODE will display the error message:

  `File` *file-name* `is not a saved search index`

- You can invoke the command **M-X Load Search Index** to load a saved search index from anywhere within NMODE. NMODE will respond with the same prompt discussed above. After you enter the file name of a saved search index NMODE will load and enter the search index.

- When you are in a directory browser containing items for files that are saved search indexes these search indexes can be loaded using directory browser commands. First you must select an item for a file containing a saved search index that you want to be loaded. Now you can load the search index by invoking either the `Browse` or the `Type-specific Load` command. NMODE will then load the search index from the indicated file and then enter the search index. These two commands can be used on grouped items to load more than one search index. If the current item is part of a group then invoking the `Browse` or `Type-specific Load` commands will load saved search indexes for each of the files in the group.

You may want to load saved search indexes during the initialization of your NMODE session. For this reason there is a programmatic method of loading saved search indexes. If you evaluate the form:

`(nmode:load-file-index-browser-db` *file-name*`)`

NMODE will load a saved file index from *file-name*. This function is used to load saved code, file search and error indexes programmatically. File index is a generic term that covers all three of these indexes. *File-Name* must be the full path name to the search index file and must end in `.sb` when loading a saved search index. If the file does not exist or is not a saved search index NMODE will display the error message discussed above.

The function `nmode:load-file-index-browser-db` can be invoked from your NMODE initialization or customization files to load search indexes when your NMODE session is started.

When saved search indexes are loaded programmatically the search indexes are not automatically entered. However, they will be displayed in the `Search Indexes` facility just like other search indexes.

Chapter 15
Error Indexes

Introduction

The error index facility provides a means of viewing the compilation errors that have been detected in source code. When a C, Pascal or Fortran source file is compiled from inside NMODE any errors detected are added to an error index. When an item in an error index is browsed you will be placed in the file at the location corresponding to that item (i.e. the location where the error was detected).

There are additional commands available when you browse an error and enter a file. These commands allow you to move to the next or previous error position in the file. They also will inform you when there are no more errors.

You can also use the HP-UX command *make* to compile several files and have all the generated errors added to an error index.

Loading the Support Code for the Error Indexes Facility

To work with and execute error index commands, you need to load the code that implements them. This can be done in two different ways. The simplest way is to browse into Additional Facilities, put the cursor on the `Error Index` item, and execute either of the `Load` or `Browse/execute` command line commands. The difference between the two is that `Browse/execute` assumes that you want to create a new error index immediately after loading.

You can also load the error index support functions by evaluating a Lisp form. If you expect to use this type of browser often, you can have the error index facility loaded automatically by adding this form to your make-nmode or initialization file (`.nmoderc` in your home directory):

```
(require "err-browser")
```

The form sets up the facility in `NMODE Root` called `Compilation Error Indexes`, which lets you access the active error indexes. While this loads the facility, you will not necessarily see any error indexes displayed on the screen when you are in the facility. You need to complete a few more steps.

Creating New Error Indexes

After having loaded the error index facility, you can now create an error index. One method is to create an empty error index first and then explicitly add files to it. To do this,

- You can browse Additional Facilities and use the `Browse/execute` command.
- You can execute **M-X create error index** from anywhere in the system.
- From the Error Index Facility or within an error index itself, you can use `Create` and its sub-command `Error-index`.

These will prompt for the name of a new error index. After you enter the name, the system will create and then enter the new error index.

At this point, you need to add items to the error index and then save it if you want to use the error index beyond the duration of your current session. To add new items to the error index, you should execute the command named `Add-files` (as described in a later section).

Another method of creation consists of creating an error index and building the items via one command. This can be done by

- Executing **M-X compile file** from within a buffer containing a C, Pascal or Fortran source file. The error index will have the same name as the file whose code was browsed.
- Using `Type-specific` in a directory browser with sub-command `File-compile` on a file with suffix `.p`, `.c`, or C.f (for source code in C, Pascal or Fortran). The error index will have the same name as the file whose code was compiled.

This last method also has a means of handling multiple files at one time. This can be done by grouping the files you desire within the directory browser. Then use the `Type-specific File-compile` with the current item being one of the files in the group. This will prompt for the name of the error index and then create the items from each grouped file.

Error Index Facility

At the top level of NMODE, you can point to and browse into the facility called `Error Indexes`. This invokes a browser that will contain items for each of the error indexes in the NMODE system. Like working with the browser for active directories, notice the implicit two-step procedure: the facility provides a browser that lets you browse into a particular error index.

The screen display might look like this:

```
Compilation Error Indexes

    acme-code



 @ Browser (Error Indexes)
Help  Browse  Group  Filter  Create  Options  Kill  Quit
```

Note that the title and mode line indicate your location in NMODE and you have a menu of commands. If you select an item in this browser and invoke the `Browse` command you will enter the corresponding error index. Each item in an error index is a pointer to where a compilation error was detected in a source code file.

An Error Index

The previous example of the Error Indexes Facility contained an item called acme-code. Pointing to and browsing into the acme-code error index takes you into that particular set of items. Each item in this database lets you access the source code file that contains the compilation error.

The screen display might look like this:

```
     Compilation Error Index: acme-code

         /users/guest/progs/demo.c

  syntax error




     @ Browser (Error Index) acme-code

Help  Browse  Group  Create  Utilities  Options  Add-files  Quit
```

Like all browsers, you see a title and mode line that indicate your location in NMODE and you have a menu of commands that are discussed later. There are actually two items in this error index. That is:

```
/users/guest/progs/demo.c

syntax error
```

are the two index items that point into the source file named demo.c. You can point to and browse any of the two lines (the file name or error message item) and NMODE will find the source code file, load it into memory, and visit the file in a buffer according to the following scheme:

- If you point to and browse /users/guest/..., NMODE locates you at the beginning of the file.
- If you point to and browse the error message, syntax error, NMODE locates at the line where the error was detected in the associated source file.

Regardless of your location within the buffer, you can edit the file as desired. Note that the above display focused on an extremely simple situation to illustrate the idea. Actually, error indexes will normally contain many items for a single file and might contain several files. In general, the list for a error index looks like this:

```
file-name-1
```

error-message-1
error-message-2

.
.
.

error-message-n

file-name-2

Executing Browser Commands (Error Indexes)

An error index browser has the following menu of commands:

```
Help  Browse  Group  Filter  Create  Utility  Options  Add-files  Quit
```

The commands named `Help`, `Browse`, `Group`, `Filter`, `Create`, and `Quit` were described earlier in the section called "Common Browser Commands" in the chapter called "Introduction to Browsers". The next few sections discuss the other commands.

The Add-files Command (Error Index)

The `Add-files` command allows you to add to the list of files that are being indexed in a particular error index. During execution, the command compiles source code files that are being added. It creates an item for each error detected. Those items appear in the error index under the name of the file containing the errors. Use the `Add-files` command when you make an error index or when you want to add more files to an existing index. Only add files that have an appropriate suffix: `.p`, `.c`, or `.f`.

The `Add-files` command prompts:

```
File (or "wildcard" spec): (relative to '/users/guest/')
```

Type the file name, or path name for source code files, that you want to add to the error index. As examples:

```
acme/code22.c
```

adds errors from one C source file, `code22.c`, from the directory named `acme` under `/users/guest/`. Using the wildcard character * allows multiple files to be specified. Entering

```
/users/joint-project/ace/*.c
```

adds errors for all the C Source files in the directory named `ace` under `/users/joint-project/`.

If you specify a directory name only, then all of the files in the directory will be added.

When files are added to an error index (or recompiled via the **Utilities Update** command discussed below), NMODE will display a message for each file showing how it is being compiled. This message has the form:

`Compiling` *file-name* `with options` *compile-options*

Compile-Options are the actual options passed to the compiler.

The Utilities Command (Error Index)

Executing `Utilities` provides the following menu of second-level commands:

```
Update  Kill  Write  Current-File  Quit-utilities
```

These sub-commands work as follows:

`Update` — Recompiles the file associated with the current item in the error index and updates the items in the error index to reflect the actual state of the source files. If you want more than one file updated, group the file name items of the desired files in the error index. Then select one of the grouped items and invoke `Utilities Update`. If the file that is being recompiled is in an unsaved buffer NMODE will display a prompt of the form:

`Save file "`*file-name*`"? (Default is: 'Yes')`

If you enter Return the buffer will be saved to the file and the new version of the file will be compiled. If you enter no the buffer will not be saved and the older version of the file will be compiled.

`Kill` — Removes the highlighted item from the error index. This only affects the error index and has no effect on the actual source code file. If the highlighted item is a file name, all items of that file and the file name will be removed.

`Write` — Saves the error index into a file. Then it can be loaded in a subsequent NMODE session. See the section "Saving and Restoring Error Indexes" later in this chapter for further details.

`Current-file`	Some source code files have more errors than can be displayed in the window. Executing `Current-file` displays in the message area the name of the source file for the current highlighted item.
`Quit-utilities`	Exits the Utilities command and returns to the error index.

Running make

Many programmers do not call the C, Pascal or Fortran compilers directly to compile source modules except for the most trivial programs. Instead they use the program *make* to control compilation. The *make* program reads a file (commonly called a make file) , which contains information on the dependencies one file may have on another. Based on the dependencies and on the modification dates of files, *make* determines what commands to execute. An error index can invoke the program *make* and add errors from compilations of C, Pascal or Fortran source done by *make* to that error index.

The command to invoke *make* is `Utilities Run-make`. When this command is executed, *make* is run with arguments determined by the values of the options for *make* in the error index. See the section "The Options Command" below for details on *make* options.

When *make* is run NMODE checks if there are any unsaved buffers containing files associated with the error index. If there are, a prompt of the following form is displayed:

```
Saved modified buffers associated with error index? (Default is: 'Yes')
```

To save the buffers enter Return. If you do not want the buffers saved enter `no`.

Next the system displays in the echo area of the screen a message of the form:

`Running make with options` *make-options*

Make-Options is the actual arguments that are passed to the *make* process.

After the *make* program has terminated the output of the *make* is processed to build error index items. Any commands executed by *make* that are compilations of C, Pascal or Fortran sources will cause error index items to be built for that file. If any errors were detected by the compiler then items for those error messages will be built and displayed in the error index below the name of the source file containing the errors. If no errors occurred then the file name will be displayed followed by two blank lines. Just prior to adding error index items that were generated by *make*, all error items in the error index are removed. Thus after running *make* only the errors generated by *make* will be in the error index.

Special Items Added to Error Index

Other commands may be invoked from a *make* file besides invocations of the compilers of C, Pascal or Fortran source files. The lines in the output from *make* that are caused by executing these other commands are also added to the error index so that a user will not lose information because *make* was run from the error index. For each non-compilation command the errors generated by that command will be displayed in the error index in the following form:

```
line containing command as executed by make
```

line containing first message from command execution

line containing last message from command execution

The messages associated with a command may not always be error messages. In some cases they may just be informative messages generated by the command.

The items associated with non-compilation commands are displayed for the user's information only and cannot be browsed. Attempting to browse one of these items will cause the system to display an error message:

```
Item has no associated source file
```

The items associated with a non-compilation command are displayed together. The groups of error index items associated with either files or non-compilation items are displayed in lexicographic order according to the first line of each group (i.e. either the full path name of the file or for non-compilation commands, the command itself).

Restrictions on Use of make in Error Index

Because of the way the error index processes the output from *make* there are certain restriction on how *make* is run and on command lines in the make file.

- No command in a make file can compile more than one source file.
- The options `-d`, `-n`, `-q` and `-s` are not sent as arguments to *make*. See the section "The Options Command" below for details on options used with *make*.

The Options Command (Error Index)

For the C, Pascal and Fortran compilers and for the program *make* there are options, which control the process of running the program that generates errors for the error index. These options may be modified with the `Options` command.

There is a chapter devoted to user options called "The Browser for User Options". That chapter discusses how to access and use options browsers in general. You may want to refer to that chapter as this section will describe options pertaining to code indexes.

Executing `Options` provides the following menu of second-level commands:

```
options:          C  Fortran  Pascal  Make   Quit-options
```

Choose the type of options you wish to customize. This will place you in an options browser for that compiler or for the *make* program. The options browser provides the following menu of commands.

```
     Help Browse/modify Group Filter Sort Write Restore-default Quit
```

The commands named `Help`, `Group`, `Filter`, `Sort`, `Write`, and `Quit` are familiar from previous chapters dealing with browsing in general. Executing `Browse/modify` lets you alter the value of the current option. Executing `Restore-default` assigns the default value to the current option.

Options for C, Pascal and Fortran Compilers

Following shows the options (system variables) for each language. Each list depicts what you see on the screen when you invoke the related user options browser. In each case, you see the assigned default values for options.

If you execute `Options` and then execute `Pascal`, you see:

```
   User Options:  Pascal Compilation Options
     Currently saved in "$LISP/config/pscl-error.opt"

  COMPILER OPTIONS:
    Compiler execute directory                  "."
    Compiler option string                      "-c"
    Compiler program name                       "pc"
```

If you execute `Options` and then execute `C`, you see:

```
   User Options:  C Compilation Options
     Currently saved in "$LISP/config/c-error.opt"

  COMPILER OPTIONS:
    Compiler execute directory                 "."
    Compiler option string                     "-c"
    Compiler program name                      "cc"
```

If you execute `Options` and then execute `Fortran`, you see:

```
   User Options:  Fortran Compilation Options
     Currently saved in "$LISP/config/ftn-error.opt"

  COMPILER OPTIONS:
    Compiler execute directory                 "."
    Compiler option(s) string                  "-c"
    Compiler program name                      "f77"
```

The options for each language have the same format and meaning.

Name of option in browser	Meaning of option value
Compiler option string	This is a string containing a list of options to the compiler. The documentation for the particular compiler should be checked for the exact format of these options. The default for this option -c tells the compiler to suppress loading of the object file after the compilation succeeds. The HP-UX reference manual has more information on the options for a particular compiler.
Compiler program name	This is a string containing the name of the program to be run to perform the compilation. It should be modified for special versions of compilers (e.g cross compilers for other machines).
Compile execute directory	This is a string containing the path name of the directory that the compilation will be executed in. If a file being compiled includes other files the path names of those files will be relative to this directory. If this option value is a relative path name then is relative to the directory containing the source file.

When a compilation is initiated (via the `Add-files` or `Utilities Update` commands), the options will be used to build a command string to tell HP-UX how to do the compilation. For each file being compiled the suffix will determine the language and the option values for that language will determine the command string. The command string will have the following form:

```
cd directory ; program option-string file-name
```

Directory is determined from the value of the `Compile execute directory` option for the particular language, *Program* is the `Compiler program name` option for the particular language and *option-string* is the `Compiler option string` option for the particular language. For example executing an `Add-files` command for the file `/users/guest/test.c` with the C options set at their default values would result in the command string

```
cd /users/guest ; cc -c test.c
```

being executed to perform the compilation.

Options for make

If you execute `Options` and then execute *make*, you see:

```
User Options:  Make Options
  Currently saved in "$LISP/config/make-error.opt"

MAKE OPTIONS:
  Make execute directory                      "."
  Make file name                              ""
  Make option string                          "-k"
  Make target                                 ""
```

These options have the following meaning:

```
Name of option in browser   Meaning of option value
-------------------------   -------------------------------------------
Make execute directory      This is a string containing the
                            directory that the system will be in
                            when executing make.

Make file name              This is a string containing the name
                            of the description file that specifies
                            the dependencies and commands that
                            make will execute.  If the value is
                            "" then make will search the
                            current directory for the default make
                            file names as described in the HP-UX
                            manual in section make(1).
```

`Make option string`	This is a string containing options for *make*. See section *make(1)* of HP-UX manual for details on *make* options.
`Make target`	This is a string containing the target goals that *make* will try to build. See *make(1)* in HP-UX documentation for further information on make targets.

When *make* is invoked from an error index (via the `Utilities Run-make` command), the options will be used to build a command string to tell HP-UX how to run *make*. The command string will have the following form:

cd *directory* ; make *make-options target*

Directory is the value of the `Make execute directory` option, and *target* is the value of the `Make target` option. *Make-Options* is obtained by removing all occurrences of the ignored options (`-d`, `-n -q` and `-s`) from the value of `Make option string`. Then if the value of `Make file name` is not "", `-f` and the file name are appended to *make-options*.

For example, if the *make* options were set as follows:

```
Make execute directory    "~/test"
Make file name            "special-make"
Make option string        "-k -q"
Make target               "install"
```

then invoking `Utilities Run-make` would result in the command string:

```
cd ~/test ; make -k -f special-make install
```

being executed to run *make*.

Browsing an Error Index Item

Browsing the current item in an error index activates an error minor mode in the buffer containing the source file. The mode line will indicate this by displaying the language of the source file and the word Error in parentheses. Additionally the error message associated with the current error index item is displayed in the prompt line.

The error minor mode provides two commands that let you move to the previous or next error without exiting to the error index. The two key sequences and commands are:

M-X next error	Positions the cursor at the next error. If there are no errors after the current one in the associated error index the error message **No next error** will be displayed. This command has an abbreviated form (**C-X M-N**).
M-X previous error	Positions the cursor at the previous error. If there are no errors before the current one in the associated error index the error message **No previous error** will be displayed. This command has an abbreviated form (**C-X M-P**).

Both commands automatically change the displayed file when the new error is in a different file. After positioning the cursor on the next or previous error, the error message associated with the new current error is displayed in the message area. These commands will not find the next or previous errors when the line at the position of the error has been changed in the buffer containing the source file. After you exit (via **C-M-L**) the source file containing errors, you will return to the error index. If you changed the current item by invoking either **M-X next error** or **M-X previous error** commands, it will be reflected in the display of the error index. The new current item will be highlighted.

Saving and Restoring Error Indexes

When an NMODE session is exited all error indexes that were created during that session are destroyed. To save the effort of having to create error indexes each time a new NMODE session is started there are commands that allow you to save an error index after it has been created and restore that error index in a subsequent session.

Saving an Error Index

To save an error index you must be in the error index you want to save. Execute `Utilities` and `Write` to save the error index. NMODE will display a prompt something like:

```
    Write browser database to file (Default is '$HOME/acme-code.eb')
```

If you enter a Return in response to the prompt the default value is used as the file name to save to. If you enter the path name of a directory followed by / then the error index is saved in the file in that directory. In the example above if you entered **/users/guest/code-indexes/** the error index would be saved in the file **/users/guest/code-indexes/acme-code.eb**. If you enter the path name to a file the system will modify the name of the file to insure that it is not longer than 14 characters and that it ends in `.eb`. The error index be will saved in the file with the resulting name. Saved error index file names will always end in the suffix `.eb` so they can be identified.

After a response is entered NMODE will display the message:

```
    Writing index database ...
```

When the save operation is complete NMODE will display a message of the form:

    File *file-name* written.

to indicate that the save operation is complete.

Loading Saved Error Indexes

An error index that has been created and saved can be loaded back into NMODE either interactively or programmatically. There are several methods that can be used to load a saved error index interactively.

- Within the `Error Indexes` facility you can invoke the `Create` command. NMODE will display a new menu in the prompt area. If you now invoke the command `Load-error-index` NMODE will display a prompt similar to:

  ```
  Load index file (type defaults to ".eb"): (relative to /users/guest/)
  ```

  You should enter the name of the file containing a saved error index. You do not need to enter the full path name of the file. The suffix can be omitted. It will default to `.eb` as the prompt `(type defaults to ".eb")` indicates. Also the path name can be entered relative to the working directory of NMODE, which is displayed after `relative to` in the prompt. After you enter a file name NMODE will load the error index saved in the file into the system and move you into that error index. If the file does not exist or is not a saved error index, NMODE will display the error message:

  `File` *file-name* `is not a saved error index`

- You can invoke the command **M-X Load Error Index** to load a saved error index from anywhere within NMODE. NMODE will respond with the same prompt discussed above. After you enter the file name of a saved error index NMODE will load and enter the error index.

- When you are in a directory browser containing items for files that are saved error indexes these error indexes can be loaded using directory browser commands. First you must select an item for a file containing a saved error index that you want to be loaded. Now you can load the error index by invoking either the `Browse` or the `Type-specific Load` command. NMODE will then load the error index from the indicated file and then enter the error index. These two commands can be used on grouped items to load more than one error index. If the current item is part of a group then invoking the `Browse` or `Type-specific Load` commands will load saved error indexes for each of the files in the group.

You may want to load saved error indexes during the initialization of your NMODE session. For this reason there is a programmatic method of loading saved error indexes. If you evaluate the form:

`(nmode:load-file-index-browser-db` *file-name*

`NMODE will load a saved file index from file-name.` This function is used to load saved code, file search and error indexes programmatically. File index is a generic term that covers all three of these indexes. *File-Name* must be the full path name to the error index file and must end in `.eb` when loading a saved error index. If the file does not exist or is not a saved error index NMODE will display the error message discussed above.

The function `nmode:load-file-index-browser-db` can be invoked from your NMODE initialization or customization files to load error indexes when your NMODE session is started.

When saved error indexes are loaded programmatically the error indexes are not automatically entered. However, they will be displayed in the `Error Indexes` facility just like other error indexes.

Chapter 16
User Options

Introduction

This chapter discusses the User Options facility which allows customization of other system facilities. User Options for a particular facility are described within the chapter for that facility, All User Options are reviewed here.

Recall that the NMODE user environment has many system variables (user options) that are customizable. This means that you can invoke a browser of user options for a particular facility and modify the values of available options, thereby manipulating the facility to suit your needs. For example, recall that the display of information about the size, write date, and permissions for items in an active directory depends on the values of the user options for the browser for directories.

From NMODE Root, the facility called `User Options` provides access to browsers for modifying specific user options, which in turn, provide quick and easy access to system variables.

User options browsers display a list of fields, specific to a certain facility, corresponding to system variables and functions. The fields are displayed with a label that identifies the option, and the current value of the option.

The next several sections discuss how to work with the options browsers.

Selecting User Options

This section discusses the user options and some things you need to do to be able to modify their values.

You can invoke a browser for user options in several ways.

- When you are using some browser that lets you set options, execute the command named `Options` (or execute **M-X Options**). This invokes a browser for user options related to the current browser and provides a menu of commands that let you manipulate the related options. This command is discussed in detail later.
- If you want to change options for more than one facility, from NMODE Root browse into `User Options`, or execute **M-X options** from anywhere in NMODE. You will then have a choice of several User Options browsers for various facilities.
- If you browse the directory `$LISP/config` (or the directory specified by (`*user-customize-prefix*`), you can point to and browse into a file that has an `.opt` suffix. This invokes a browser for user options for the facility implied by the filename. For example, browsing into `lisp-vars.opt` invokes an options browser for changing the values of some Common Lisp system variables.
- Execute **M-X load options**: This command prompts for a user options file name and either: creates a browser for user options for the specified file when no such browser exists; or reenters the specified browser.

However you first enter an options browser, the browser is added to the facility called `User Options`.

The User Options Facility

This section describes the facility called `User Options`, which can be browsed from NMODE Root. Depending on what facilities are loaded, the display looks like this:

```
User Options

   NMODE Window Creation Options
   Execution Monitor
   Debugger Printing
   Stack Item Visibility Options
   HP-UX Access Options
   Common Lisp User Options
   Directory Options
   NMODE General User Options



 @  Browser (User Options)

Help  Browse  Group  Filter  Create  Options  Kill  Quit
```

Your list of items can vary considerably, depending on how you customized your initialization file and which options browsers you have used during the current NMODE session and any additional facilities you have used. The item named `NMODE General User Options` is special; this item is described later in the section called "Options".

Notice the familiar menu of commands. The commands named `Help`, `Browse`, `Group`, `Filter`, `Kill`, and `Quit` work according to directions given earlier in the section called "Common Browser Commands" in the "Introduction to Browsers" chapter. The command named `Create` is discussed next, and `Options` is described in the next section.

Executing `Create` provides the following menu of second level commands.

```
Load-options-file  File  Buffer  Other  Quit-create
```

The `Load-options-file` performs the same function as the **M-X Load Options** command discussed in the previous section.

The other `Create` second-level commands, `File`, `Buffer`, and `Quit-create`, work as their name implies. Executing `Other` exits the current browser and enters a browser for creating other things. This was explained earlier in the chapter called "Introduction to Browsers".

Loading User Options

Some options are always loaded. You can examine these options by browsing the `User Options` facility at `NMODE Root`.

```
User Options

  NMODE Window Creation Options
  Common Lisp User Options
  NMODE General User Options
  Directory Options
```

Generally, a new Option will appear in this list when one of the facilities appearing in `Additional Facilities` is loaded. When you load one of the facilities, NMODE checks the value of `*user-customize-prefix*` to determine the directory name to check for the options file.

When you modify the options for a facility, you can save the modified options in the directory of your choice (defaults to the value of `nmode:*user-customize-prefix*`). The next time the facility is loaded, the modified options are restored from the directory specified by `*user-customize-prefix*`. If the desired options are not found in the location specified by `*user-customize-prefix*`, then they are read from the `$LISP/config/` directory, which contains the original versions of the options files.

Options Browser Command Menu

The Help, Group, Filter, and Quit commands were discussed in the "Introduction to Browsers" chapter of this manual.

Browse/modify

The **Browse/modify** command allows you to change values or invoke functions. Here are the different types of options and how to change them.

`<Browse to invoke function>`	Simply Browse this item to invoke the function indicated by the description. For example, the HP-UX Access facility will create an HP-UX shell if you browse the `Create System Shell only` function.
Yes/No	Many item are simply "yes" and "no" toggles. For instance, in the Directory Options browser, you can have the file size displayed in a directory browser. By browsing this item, you can control whether the file size is shown.

T/NIL	Similar to the "yes" and "no" toggle described above. See the `Common Lisp User Options` for examples. At other times a `NIL` may indicate an unset option, in which case you will be prompted for a value.
Multiple Choice	Some items provide three or more choices. Repeated browsing cycles through all possible choices. For example, the `Common Lisp User Options` item called: `*print-case*` allows printing to be in uppercase, lowercase, or capitalized.
Numeric	Browsing a numeric item prompts you to enter a new value. An example is the `NMODE General User Options`. Browsing `Undo Stack Depth for New Buffers` prompts you for the number of changes to be retained during an editing session. You may enter a new number or accept the default value by pressing Return.
String	Many items accept a string as input. Browsing a string option prompts you to enter a string. It is important to note the default value of a string can be the null string ("") and pressing Return will result in the value being set to the null string. For example, in the `NMODE General User Options`, the `Author name` item will prompt you to enter your name. **C-G** will abort the input without changing the value of the option.

Sort
The Sort command allows you to rearrange the list of options alphabetically or by argument type.

Write
Once you have modified the options to a particular facility, use the Write command to save your changes. You are prompted for a filename. The default value is the pathname specified by `*user-customize-prefix*` in your `.nmoderc` file (set to `$HOME` by default) plus the name of the Options file. Do not change the base file name of an options file. If you do, NMODE will not load it automatically in future sessions.

By allowing the Options file to be written to each users "home" directory, each user can have his or her own customizations.

Once the file is written, the next time NMODE is executed the modified options will be in effect.

Restore-default

At some time you may wish to restore an option to its factory default. The Restore-default command allows you to restore the currently selected option to its original value. This is possible because each Options Browser retains a copy of the factory default option values.

Changing more than one option is possible by using an argument (such as **C-U 5** to change the next five options) or by grouping.

A Typical Options Menu

Browsing `User Options` at the top level, and then browsing `NMODE General User Options` results in the following display:

```
User Options:  NMODE General User Options
  Currently saved in  $LISP/config/nmode.opt

NMODE ENVIRONMENT CONTROL:
  Multiple selection of Group (>) items:     Yes
  HP-UX Command for Nmode Printing           "lp"
  Symbol for Selected Window                 " @ "
  Undo Stack Depth for New Buffers           50
  Expose OUTPUT Buffer for any output        Yes
  Add terminating newline to buffer          Yes
  Set Up NKeys                               <Browse to invoke function>

FILE SYSTEM CONTROL:
  Merge Default File Name Suffixes           No
  Directory for LAN Network's Special File   "/net/"

BUFFER & FILE HEADER CONTROL:
  User Name for Revisions                    "Ed Jones"
  Author Name                                "Ed Jones, Ace Corp."
  Initial Status                             "Experimental (Do Not Dist.)"
  User Company Name for Copyright            "Ace Corp."
  Update Revision Date on Writes             Yes
  Case-sensitivity on     No


 @  Browser (User Options) NMODE General User Options

Help  Browse/modify  Group  Filter  Sort  Write  Restore-default  Quit
```

The fields on the right show the current values for the options named on the left. Note that the values control various aspects of your environment. For example, with the above value of the `Undo Stack Depth for New Buffers` option, you will be able to undo the last 50 changes in a buffer with the `Undo` command. As another example, the values assigned to options for File Header Control will show up in file headers that you insert with **M-X Make Header**.

An options browser lets you manipulate user options for a particular facility. What changes from one options browser to another is the fields, options, and values that you can manipulate.

Available Options

If you are familiar with NMODE and just want to explore some options browsers on your own, the directory named `$LISP/config` contains the following option files.

```
c-code.opt
c-edit.opt
c-error.opt
db-print.opt
directory.opt
ftn-code.opt
ftn-edit.opt
ftn-error.opt
hp-ux.opt
lisp-code.opt
lisp-vars.opt
make-error.opt
nmode.opt
pscl-code.opt
pscl-edit.opt
pscl-error.opt
stack-vis.opt
window.opt
xm-options.opt
```

Pointing to and browsing into any such file invokes an options browser for the facility implied by the filename.

Options

Here is a synopsis of each of the Options Browsers.

NMODE Window Creation Options: Contains alterable specifications for the next window to be created as well as a locator device enable and disable function.

Common Lisp User Options: Contains alterable lisp variables that control reading and printing options as well as error system control and optimization control.

Directory Options: Provides display control on directory items as well as additional control on how files are read in and manipulated.

NMODE General User Options: Provides control for the basic nmode environment including file system and file header control file header control.

Make Options: Options for controlling the running of make files.

Pascal Compilation Options: Options for controlling the compilation of pascal files.

Fortran Compilation Options: Options for controlling the compilation of fortran files.

C Compilation Options: Options for controlling the compilation of C files.

LISP Code Index Options: Provides control over what code items (lines of code, when browsed, enters the file at that location in the code) are extracted based on lisp constructs and column location of forms.

Pascal Code Index Options: Provides control over what code items (lines of code, when browsed, enters the file at that location in the code) are extracted based on Pascal constructs.

Fortran Code Index Options: Provides control over what code items (lines of code, when browsed, enters the file at that location in the code) are extracted based on Fortran constructs.

C Code Index Options: Provides control over what code items (lines of code, when browsed, enters the file at that location in the code) are extracted based on C constructs.

Execution Monitor: Provides control over the basic functioning of the execution monitor as well the levels to which it executes.

Debugger Printing: Provides control over how much information is printed by the execution stack analyzer and the execution monitor.

Stack Item Visibility Options: Controls what functions are shown in the stack analyzer.

HP-UX Access Options: Provides control over creation and killing of the System Shell and shell-buffers as well as options for the popup menus for HP-UX commands.

Fortran Edit Options: Provides program formatting options.

Pascal Edit Options: Provides program formatting options.

C Edit Options: Provides program formatting options as well as a cursor bounce option.

Chapter 17 Customization

Introduction

The NMODE user environment can be changed to fit the way you work. This chapter describes how to make various changes.

You can change the following things.

- What a particular keystroke does.
- The labels and functions of softkeys.
- System variables (options) that change the actions of the environment.

At a more comprehensive level, you can register new applications in the Applications Browser and write your own functions to be used as editor commands.

How Customizations are Made

Since NMODE is written in Lisp, you can access and use functions that change its behavior. Keep in mind that although these functions are available, they are not part of Common Lisp. Instead, the functions that implement the NMODE user environment are in the `nmode` package.

A customization is usually one or more function calls that you originate. This chapter and the *NMODE Command Reference* contain the names of the functions usually used to make customizations. These functions, combined with system variables that can be accessed and changed, let you modify the NMODE environment so that it meets your needs.

When Customizations are Made

Customizations can be made at two times.

- The initialization file in your home directory called `.nmoderc` contains forms that are evaluated at the beginning of each invocation of NMODE. By inserting or uncommenting forms in this file, you can customize NMODE each time it is invoked.
- Options are loaded from the `nmode:*user-customize-prefix*` directory after initialization files are read.
- Customizations can be made "on the fly" during any session with NMODE, but such customizations are not persistent. They last only for the duration of the particular session.

There is also a file named `$LISP/config/nmode.init` which contains customizations that you can make for every user of the system. You should set up this file to suit your needs before you invoke NMODE so that the environment reflects the customizations you want. Every user's `.nmoderc` should include the form:

```
(nmode:nmode-read-and-evaluate-file nmode:nmode-default-init-file-name)
```

which puts the customizations in the file named `$LISP/config/nmode.init` into effect.

Where to Start

Reading this chapter should give you a good idea of what types of customizations are possible and how to make them. Another good source is the example configuration file called `$LISP/config/nmoderc`, which is shipped with your system. This file contains forms for many typical modifications of your environment, along with comments that explain how to change the forms so they suit your needs. The global initialization file named `$LISP/config/nmode.init` also contains examples.

Key Mappings

When you are in NMODE and press a key, possibly in combination with zero or more of the keys Shift, Meta (left Extend char), ESC, or CTRL, you invoke a function that performs a task. The association between a **key sequence** and a **function** is known as a **key binding** or **key mapping**. The combination mentioned **zero** or more keys because you can alter the binding for a single key. You should do this with caution and make sure you do not alter the binding of a single key (e.g. Return) that performs some fundamental task in the system.

Tables known as **command lists** store the bindings for key sequences, and are associated with modes. For example, when you edit in Lisp Mode, the key sequence **C-M-Q** is bound to the NMODE function `lisp-indent-sexpr`. On the other hand, in Text mode, **C-M-Q** is not bound to any function and pressing that key sequence produces an error message.

Determining Key Mappings

To determine what function is bound to a key sequence, execute the Help Binding command, **M-/** or **M-?**. The command prompts you to type the key sequence or extended command you are interested in, and then briefly displays in the echo area the name of the function to which that key sequence is bound. If the key sequence is unbound, the command displays the message, `Function: Undefined`. This command even works for new key bindings you might create. In general, you should create key bindings for situations that have no current binding. That way, you can add functionality without detracting from existing functionality. If you need to know which bindings are in effect in a particular mode, you can get into the desired mode, execute **M-X help bindings** and select the `All` option. You will be given a choice of where to display the list of bindings.

Changing Key Mappings

Suppose you want to make the key sequence **M-=** set a mark. That is, you want to bind **M-=** to `set-mark-command`, which is the function that actually executes the Set Mark command. The implied distinctions are important because, in NMODE, you have:

- The name of a command; for example, Grow Pane command.
- The name of the function that executes the command; for example, `grow-window-command`.
- The arbitrary key sequence or extended command which accesses the function; for example, **C-X ^** or **M-X grow pane up**, respectively.

There is usually a close relationship between the name of the command and the name of the function, but not always. The key sequence is rather arbitrary, but NMODE does have some conventions concerning the Control, Meta, and Control-Meta keys for assigning key bindings. For example, deleting a line in any editing mode is **C-K**, deleting a sentence in Text Mode is **M-K**, and deleting a form in Lisp Mode is **C-M-K** where **K** is used each time to denote the deleting (Killing), and the prefix characters progress from **Control** to **Meta** to **Control-Meta**.

To set up a key binding, or to alter any other key binding, you must:

1. Determine what function performs the action you want;
2. Choose a key sequence that you want to bind to the function (preferably one that has no default binding);
3. Bind the key sequence to the function.

Each step is discussed next.

Determining the Appropriate Function

If you do not know the name of the command that performs the task you want, execute **M-X Explain**, which prompts you to enter a string. Enter a string such as `region` or whatever relates to the type of commands you want. Examine the list of command names and descriptions, and identify the command you need. You can browse an item in the list to get more information, such as the name of the Lisp function that implements the command. Doing this with set mark, for example, reveals that it is implemented by the function, `set-mark-command`. The names of functions that implement commands can also be found in the *NMODE Command Reference* manual.

Choosing a Key Sequence

Finding an unbound key sequence is a matter of knowing what keys have default bindings, and trial and error. You can print out a list of all bindings with the NMODE Help facility's List command. You can also scan the lists of NMODE commands in Appendix A, which shows the bindings for commands according to logical categories. Doing this reveals that there is no default binding for **M-=**.

Creating a Key Mapping

How you perform this step depends on whether or not you want to have your new binding persist beyond your current session. If you only need a binding for the current session, use the **M-X Set Key** command. It prompts you for the function and then the key sequence. Enter the function name (no parens) and type the key sequence like you would to execute the command. The function must have no arguments.

To have your customizations in effect every time you use NMODE, you need to add forms to your file named `.nmoderc`. Several existing functions help you customize the key sequences used to access command functions. One of them is:

(**nmode:add-to-command-list** *command-list key-sequence function-name*) *Function*

The function **nmode:add-to-command-list** is essentially a non-interactive **M-X Set Key**. It adds a binding of *key-sequence* and *function-name* to *command-list*. There are several command lists that you will be concerned with when making customizations, but for now we'll just use **nmode:text-command-list**.

Representing Key Sequences

Since **add-to-command-list** takes an argument called *key-sequence*, you need a way of specifying a key sequence that may include modifiers such as Control and Meta. There are three macros for doing this.

(**nmode:x-char** *key*) *Macro*
(**nmode:x-chars** {*key*}*) *Macro*
(**nmode:x-sfk** *key*) *Macro*

These three macros let you specify a key sequence in a call to **add-to-command-list**. The key sequence you are trying to represent determines which one of the three to use. The figure below shows the keyboard divided into several zones.

If the key you want to represent (possibly in combination with Control or Meta) is in zone A, then call `x-char` with the appropriate character optionally preceded by `C-` and/or `M-`. For DEL, ESC, and the space bar, use `DEL`, `ESC`, and `Space`, respectively. For letters, case is insignificant.

```
(nmode:x-char C-#)
(nmode:x-char F)
(nmode:x-char C-M-ESC)
(nmode:x-char M-j)
```

If the key you want to represent is in zone B, use `x-char` in the same way, except precede the character with `N-` (for numeric pad).

```
(nmode:x-char C-N-*)
(nmode:x-char N-7)
(nmode:x-char C-M-N-\,)   ; Need to escape the comma
```

The macro `x-chars` is used to represent sequences of the characters that `x-char` can represent. The first *key* in a call to `x-chars` should be a standard prefix character such as `C-X` or `C-]`. Having more than two *keys* in a call to `x-chars` is allowed, but does not make much sense in the context of `add-to-command-list`.

The keys that are not in either zone A or zone B can be represented with a call to `x-sfk`. The characters that are valid when preceded by zero or more of `C-`, `M-`, or `S-` (shift) are

`Reset`	`Break`	`Stop`	`Menu`
`User`	`System`	`Clear-line`	`Clear-display`
`Back-space`	`Insert-line`	`Delete-line`	`Tab`
`Insert-char`	`Delete-char`	`Caps`	`Return`
`Home`	`Prev`	`Next`	`Select`
`Up-arrow`	`Down-arrow`	`Right-arrow`	`Left-arrow`
`Enter`	`Print`	`N-Enter`	`N-Tab`

Additionally the softkeys can be specified in a call to `x-sfk` with `f1` to `f12` (the eight normal softkeys plus the four unlabelled keys in the top right corner of the keyboard).

```
(nmode:x-sfk M-Back-space)
(nmode:x-sfk Print)
(nmode:x-sfk M-f7)
(nmode:x-sfk C-M-N-Enter)
(nmode:x-sfk C-S-f9)
```

Example

To bind the keystroke **M-=** to nmode:set-mark-command every time you invoke NMODE, add the following lines to your .nmoderc.

```
(nmode:add-to-command-list 'nmode:text-command-list (nmode:x-char M-=)
     'nmode:set-mark-command)
(nmode:nmode-establish-current-mode)
```

The first line adds the binding to the command list. The second line puts that binding into effect. When making more than one key binding, you only need one call to nmode-establish-current-mode following all of the calls to add-to-command-list. Remember that to make the new customizations work immediately after adding them to .nmoderc, the forms must be evaluated.

Command Lists

Command lists are some of the underlying data structures of the editor. You do not need to know the details of how they work to make customizations, but you need to know which command list to use when you add a key binding to your system. The command list you add a key mapping to determines where that new key mapping can be used. There are six command lists that you need to know about.

basic-command-list	Any key mappings added to this command list are available anywhere in the environment (unless overridden by a more specific command list).
text-command-list	Key mappings added to this command list are available in the Emacs major mode.
mx-text-command-list	Key mappings in this command list are available in the Emacs major mode. Only **M-X** commands should be added to this list.
read-only-text-command-list	Key mappings added to this command list are available in the Emacs major mode. The functions that implement the commands on this list should not change the contents of the buffer being edited.
read-only-mx-text-command-list	Key mappings added to this command list are available in the Emacs major mode. The commands on this list should be **M-X** commands that do not change the contents of the buffer being edited.
lisp-command-list	The key mappings in this command list are available in the Lisp minor mode.
text-language-command-list	Key mappings in this list are available in Text minor mode.

`user-keys-command-list`	This is the command list used by **M-X set key** and named command macros. After adding a command to this list, the parameterless function `nmode::activate-user-keys-mode` should be called.

As an example of when command lists are important, imagine that you often switch between the Lisp and Text minor modes, and you're tired of having to type in the M-X command. You decide that you want **M-P** to switch you from Text mode to Lisp mode, and vice versa. In Text mode, the function to get to Lisp mode is `lisp-mode-command`. In Lisp mode, the function to get to Text mode is `text-mode-command`.

```
 (nmode:add-to-command-list 'nmode:text-command-list (nmode:x-char M-P)
'nmode:lisp-mode-command)
 (nmode:add-to-command-list 'nmode:lisp-command-list (nmode:x-char M-P)
'nmode:text-mode-command)
 (nmode:nmode-establish-current-mode)
```

The first form makes **M-P** invoke the function `lisp-mode-command` whenever you're in the Emacs major mode. The second form overrides that binding whenever you're in the Lisp minor mode to make **M-P** invoke `text-mode-command`.

This customization is not possible with the **M-X Set Key** command, because **M-X Set Key** only deals with the `user-keys-command-list`. We could do a similar customization by setting **M-P** to `lisp-mode-command`, but this would only toggle the Lisp minor mode, not go directly between Lisp and Text mode.

Adding M-X Commands

You add a M-X command to a command list in the same way, but with a slightly different syntax for specifying the key sequence. Suppose you wanted to bind the function `browser-browser-command` to **M-X Let me at them browsers**. The following forms would do the job:

```
(nmode:add-to-command-list 'nmode:basic-command-list
     (nmode:m-x "Let me at them browsers")
'nmode:browser-browser-command)
(nmode:nmode-establish-current-mode)
```

There is no way to add M-X commands with **M-X Set Key**. All extended command completion features will work with extended commands that you add.

Removing Key Mappings

The function `remove-from-command-list` takes key mappings off command lists. Here is the basic function that does the work:

```
(nmode:remove-from-command-list command-list key-sequence)
```

If you decided you didn't like your **M-P** command for switching between Lisp and Text mode, you could remove it with

```
(nmode:remove-from-command-list 'nmode:text-command-list (nmode:x-char M-P))
(nmode:remove-from-command-list 'nmode:lisp-command-list (nmode:x-char M-P))
(nmode:nmode-establish-current-mode)
```

Since bindings you add with **M-X Set Key** are added to the `user-keys-command-list`, remove them from the system with

```
(nmode:remove-from-command-list 'nmode::user-keys-command-list key-sequence)
```

Writing Command Functions

Since NMODE commands are implemented as functions, and you can choose the name of the function a particular key sequence invokes, you can write your own NMODE commands.

Combining Existing Commands

If you find yourself often repeating a series of commands, you may want to write a function that executes the commands for you. For instance, if you're editing text, and sometimes you want to indent blocks of text differently (like for long passages taken from another document), it would be handy to have a command to do this for you, rather than having to do it by hand. The following function would produce the desired action.

```
    ;;;
    ;;; Special-indent-command indents a paragraph on both
    ;;; sides
    ;;;
    (defun special-indent-command ()    ; Command functions have no parameters
      (let ((old-fill-column nmode:fill-column)
     (old-left-fill-column nmode:left-fill-column)
     (old-paragraph-indent nmode:paragraph-indent))
(setf nmode:fill-column 60
      nmode:left-fill-column 10
      nmode:paragraph-indent 0)
(nmode:fill-paragraph-command)
(setf nmode:fill-column old-fill-column
      nmode:left-fill-column old-left-fill-column
      nmode:paragraph-indent old-paragraph-indent)
)
      )
```

After defining this function, bind it to a key sequence using:

```
(nmode:add-to-command-list 'nmode:text-command-list (nmode:x-char C-M-Q)
'special-indent-command)
(nmode:nmode-establish-current-mode)
```

You can find the names of functions you might want to use to create your own commands with the Show Binding command, M-/, or by looking in the *NMODE Command Reference*. If you define a lot of command functions, put all the definitions in one file, compile it, and then load that file in your .nmoderc before making the key bindings.

Softkeys

NMODE lets you change the labels and functions of the softkeys, as well as providing a default softkey setup.

The function `nmode-establish-softkey` is provided to let you define your own user softkeys. The basic form is:

```
(nmode:nmode-establish-softkey key function-name label-string)
```

Key should be a call to `x-sfk` specifying the softkey you wish to define. For instance, to specify the softkey $\boxed{\text{f1}}$ shifted, use `(nmode:x-sfk S-f1)`. *Function-name* is the name of the function that you want this softkey to invoke, and *label-string* is the string that will appear in the softkey label. To use this capability, define a function that establishes your softkeys and then bind a key sequence to that function. Then, when you want your softkeys, execute that key sequence. The $\boxed{\text{User}}$ key is a good key to bind to the function that sets up your softkeys.

Here's what the section in your file named `.nmoderc` that sets up softkeys might look like.

```
(defun set-up-them-softkeys ()
  (nmode:nmode-establish-softkey (nmode:x-sfk f1)
     'nmode:execute-form-command
     "Eval Form")
  (nmode:nmode-establish-softkey (nmode:x-sfk f2)
     'nmode:move-backward-defun-command
     "Back Form")
  (nmode:nmode-establish-softkey (nmode:x-sfk f3)
     'nmode:end-of-defun-command
     "Forward Form")
  (nmode:nmode-establish-softkey (nmode:x-sfk f4)
     'nmode:kill-forward-form-command
     "Kill Form")
  )

(nmode:add-to-command-list 'nmode:text-command-list (nmode:x-sfk USER)
'set-up-them-softkeys)
(nmode:nmode-establish-current-mode)
```

After this, you can activate your softkey definitions by pressing $\boxed{\text{User}}$. You can also change softkeys "on the fly" by simply evaluating a call to `nmode:nmode-establish-softkey`.

There is also a mode-sensitive softkey module called `nkeys`. See the example NMODE initialization file (`$LISP/config/nmoderc`) for details on how to use it.

Environment Variables

The NMODE environment contains many variables whose values affect how the system behaves. These allow you to make changes to your environment simply by changing the value of the variable. For example, the variables `nmode:fill-column` and `nmode:paragraph-indent` control what a paragraph looks like when you use the command that fills paragraphs.

The important environment variables can be changed in an options browser. Browse into `User Options` from NMODE Root to see what options browsers are available. See the chapter "User Options" for more information.

Registering Applications

The User Application browser provides a convenient mechanism for accessing Lisp application programs. You can create the browser and add an application to it with a form like:

> `(nmode:nmode-register-application` *function display-string module-name*`)`

Function is the parameterless function that you call to invoke the application; *display-string* is a string that identifies the application in the browser; and *module-name* is a string that is the name of a module that can be used as a single argument to `require` (i.e. *module* should be the name of a file in `$LISP/modules/local` minus the `.b` suffix). If the function to be called is in a package that may not exist until the application is loaded, you must precede the call to nmode-register-application with a call to make-package, and qualify the function name with "pkg::". For example,

```
(make-package "EXSYS")
(nmode:nmode-register-application
 'exsys::solve-worlds-problems    ; parameter-less function called by browse
 "P Solver"                       ; string displayed in browser line
 "app-module")                    ; module name for require
```

Appendix A
NMODE Quick Reference

Use this lift-out appendix for quick reference to the key sequences and Meta-X Command names that execute NMODE commands in a default user environment; that is, in an environment that you did not extensively customize or extend. In the three-column format for documenting commands:

- The **left column** indicates what the command does.
- The **middle column** is a key sequence that executes the command.
- The **right column** is the Meta-X prefix and English command name that executes the command.
- An **n.a.d.** in the middle column means the command has no assigned default key sequence.

The execution of some commands requires prior loading of certain facilities, which this appendix assumes are present. See the appropriate chapter in the *NMODE User's Guide* if you need information about whether a facility must be loaded to use a command. The *Installation and Overview* manual also describes the loading of facilities.

The following manuals or appendices contain additional information about executing NMODE commands.

- The *NMODE Command Reference* manual contains reference information about NMODE commands. It lists commands by name and classifies the commands according to the same categories used in this appendix. Information includes: the command name with a description, the default key sequence, the M-X Command name, the function name, the available modes, the procedure, and any applicable comments.
- *Appendix B* is a liftout card that describes the special commands that appear in the message area in the menus of browsers. These commands are also described in the chapters in the *NMODE User's Guide* that discuss assorted browsers.
- *Appendix C* is a liftout card that describes the softkeys and softkey labels available for executing selected NMODE commands.

There is no appendix that contains available popup menus, provided you have an operational mouse. The use of a mouse to execute commands is discussed in the *NMODE User's Guide.*

Options for Executing Commands

NMODE provides assorted options for executing commands. This appendix describes how to use key sequences and M-X Command names because these means are available with or without window management software and a pointing device (mouse).

Key Sequences

Executing a command by pressing a sequence of keys (a sequence of keystrokes called a key sequence) is efficient and direct, once you learn the key sequences. The major types of key sequences include: **C-<char>**, **C-X <char>**, **C-X C-<char>**, **C-M-<char>**, and **M-<char>** where **<char>** is a printing character (e.g. A-Z, 0-9, and such as %). Within the HP-UX Access facility, key sequences involve **M-H <char>** and **M-H C-<char>**. See Chapter 2 in the *Nmode User's Guide* if you need detailed information about key sequences.

A key sequence such as **C-<char>** means hold down the Control key (which is CTRL), press a character key, and release both keys. The **M-<char>** key sequence is the same, except that **M** represents the Meta key, which is the left Extend char on your keyboard. Just remember that holding down the Meta key, denoted in all documentation as **M-**, means that you hold down Extend char, which sits just left of the spacebar. Use the following conventions to execute key sequence commands.

C-X <char>	Hold down the Control key, press X, and release them. Then, press a character key.
C-X C-<char>	Hold down the Control key, press X, and release them. Then, hold down the Control key, press a character key, and release them.
C-M-<char>	Hold down the Control key, press the Meta key, and release them. Then, press a character key. The Control-Meta prefix is bound to **C-Z** for convenience (i.e. **C-Z <char>** is the same as **C-M-<char>**).
M-<char>	Hold down the Meta key, press a character key, and release them.
M-H <char>	Hold down the Meta key and press H, release them, and press a character key.
M-H C-<char>	Hold down the Meta key, press H, and release them. Then, hold down the Control key, press a character key, and release them.

Although you might encounter other key sequences, the above information illustrates the process.

M-X Command Names

A Meta prefix and Command name, **M-X <name of command>**, provides an English context for executing an NMODE command. For example, in Lisp mode within Emacs mode, executing **M-X execute form** lets you enter and evaluate a Lisp form. See Chapter 2 in the *NMODE User's Guide* for details related to executing commands via a Meta prefix and command name. Case is not important in the command name; type upper or lower-case letters, whichever is convenient.

Using Prefixed Arguments

Selected commands in the following lists were prefixed by the term, **<arg>**, to suggest using a prefixed argument when it is helpful to execute a command many times. The prefixed argument is supplied by executing the Universal Argument command, **C-U <arg>**, which is described fully in the chapter called *Basic Use of NMODE* in the *NMODE User's Guide*.

The NMODE Commands

This section lists commands alphabetically within categories. There is some redundancy in the lists to help you find commands that can fit into several categories.

Prefixes

NMODE uses the following prefixes in key sequences for commands.

Control-Meta prefix	C-Z	None available
Control-X prefix	C-X	None available
Command name prefix	M-X or C-M-X	None available
HP-UX prefix	M-H	None available
Lisp prefix	C-]	None available
NMODE help prefix	C-? or C-/	None available

Exits, Restarts, and Returns

These commands get you out of a state or system and into another state or system.

Exits

Cancel command	C-G	M-X cancel
Exit NMODE to HP-UX	C-X Z	M-X exit to HP-UX
Exit NMODE to Lisp	C-] L	M-X exit to lisp
Leave tutorial	n.a.d.	M-X leave tutorial
Lisp abort	C-] A	M-X lisp abort
Lisp quit	C-] Q	M-X lisp quit

Booting NMODE

Boot NMODE from HP-UX	Type: /lisp/bin/nmode and press Return (Note: not an NMODE command)

HP-UX Access Facility

These commands let you interface with HP-UX from NMODE. In the *NMODE User's Guide* and the *NMODE Command Reference* manual, the commands are divided into three sections so you can get detailed information about applicability and utilization of the commands. Here, they are presented alphabetically.

Complete filename	ESCAPE	M-X HP-UX complete filename
(in shell-buffer only)		
Create default HP-UX shells	n.a.d.	M-X create default HP-UX shells
Create default shell-buffer	n.a.d.	M-X create default shell-buffer
Create default system shell	n.a.d.	M-X create default system shell
Create HP-UX shell set	n.a.d.	M-X create HP-UX shell set
Create a shell buffer	n.a.d.	M-X create shell-buffer
Create the system shell	n.a.d.	M-X create system shell
Date	n.a.d.	None available
(popup only)		
Execute buffer contents	M-H B	M-X HP-UX execute buffer
Execute an HP-UX command	C-X H	M-X HP-UX execute command
Execute a line	M-H E	M-X HP-UX execute line
(in shell-buffer)	Return	
Execute and delete a line	C-M-Return	M-X HP-UX execute and delete
(in shell-buffer only)		
Execute a region	M-H R	M-X HP-UX execute region
Execute to end	M-H N	M-X HP-UX execute to end
(in shell-buffer only)	or M-H C-N	
	or S-Return	
Filter a region	M-H F	M-X HP-UX filter region
	or M-H C-F	
HP-UX options browser	n.a.d.	M-X HP-UX access options
Kill all HP-UX shells	n.a.d.	M-X kill all HP-UX shells
Kill current shell-buffer	n.a.d.	M-X kill current shell-buffer
Kill specified shell-buffer	n.a.d.	M-X kill shell-buffer
Kill the system shell	n.a.d.	M-X kill system shell
List a directory	n.a.d.	None available
(popup menu only)		
Mail	n.a.d.	None available
(popup menu only)		
Process status	n.a.d.	None available
(popup menu only)		
Recover a shell-buffer	n.a.d	M-X recover shell-buffer shell
Reference manual	n.a.d.	None available
(popup menu only)		
Script buffer	M-H M-B	M-X HP-UX script buffer
Script region	M-H M-B	M-X HP-UX script region
Send buffer	M-H C-B	M-X HP-UX send buffer
Send character	C-Return	M-X HP-UX send character
(shell-buffer only)		
Send EOF	M-H C-D	M-X HP-UX send eof
Send interrupt	M-H C-C	M-X HP-UX send interrupt

Send region	M-H S	M-X HP-UX send region
Send signal	M-H C-K	M-X HP-UX send signal
Set output buffer	M-H C-O	M-X HP-UX set output buffer
(everywhere but shell-buffer)		
Show current directory	n.a.d.	None available
(popup menu only)		
Start output from command	M-H C-Q	M-X HP-UX start output
Stop output from command	M-H C-S	M-X HP-UX stop output
Yank last HP-UX output	M-H Y	M-X HP-UX yank last output
	or M-H C-Y	

Help and Information

These commands provide information. Use them according to the type of information you need.

Apropos	n.a.d.	M-X apropos
Browse a directory	n.a.d.	M-X browse directory
Count number of strings	n.a.d.	M-X count occurrences
Dired	C-X D	M-X dired
Dispatch table to buffer	C-? K A B	None available
Dispatch table to file	C-? K A F	None available
Dispatch table to printer	C-? K A P	None available
Dispatch table to screen	C-? K A S	None available
(Can use C-/ as alternative to C-?)		
Explain	M-,	M-X explain
Find a directory	C-X C-D	M-X find directory
	or C-X D	M-X dired
Find item (need Code Br.)	M-.	M-X find item
Help with a binding	n.a.d.	M-X help binding
Invoke doc. facility	n.a.d.	M-X documentation
		or M-X help documentation
Invoke the Help facility	C-?	M-X help
(provides a menu)	or C-/	
Invoke the Tutor	C-? T	M-X tutor
Lisp help (in Lisp Mode)	C-] ?	M-X lisp help
Show fill values	n.a.d.	M-X show fill values
Show a function binding	C-? K N	M-X show function binding
Show a key binding	M-/ or M-?	M-X show key binding
What is cursor position	C-X =	M-X what cursor position
Where am I	C-? W	M-X where am i

Modes

On first execution, these commands invoke the implied major or minor mode. For Auto Fill Mode, a second execution terminates filling. For the other modes, a second execution does nothing.

Auto fill mode	n.a.d.	M-X auto fill mode
C mode	n.a.d.	M-X c mode
Emacs mode	n.a.d.	M-X emacs mode
Fortran mode	n.a.d.	M-X fortran mode
Lisp mode	n.a.d.	M-X lisp mode
Pascal mode	n.a.d.	M-X pascal mode
Reset the user bindings	n.a.d.	M-X reset user bindings
Text mode	n.a.d.	M-X text mode
User keys mode	n.a.d.	M-X user keys mode

Screens, Windows, and Panes

These commands establish or alter the state of screens, windows, and panes.

Create the default window		n.a.d.	M-X create default window
Create a lower pane	<arg>	C-X 2	M-X create lower pane
Create a side pane	<arg>	C-M-2	M-X create side pane
Create a window		n.a.d.	M-X create window
Display the next page	<arg>	Next key	M-X display next page
Display the previous pg	<arg>	Prev key	M-X display previous page
Exchange the panes		C-X E	M-X transpose panes
Full refresh of display		Clear dis.	M-X full refresh
Grow pane	<arg>	C-X ^	M-X grow pane up
Grow pane down		n.a.d.	M-X grow pane down
Grow pane left	<arg>	C-M-<	M-X grow pane left
Grow pane right	<arg>	C-M->	M-X grow pane right
Kill the pane		C-X C-K	M-X kill pane
Kill the window		n.a.d.	M-X kill window
Mark the pane		n.a.d.	M-X mark pane for transpose
Move down a line	<arg>	Down-arrow	None available
Move to pane edge	<arg>	M-R	M-X move to pane edge
Move up a line	<arg>	Up-arrow	None available
Next page	<arg>	C-V	None available
Nmode invert video		C-X V	M-X invert video
Previous page	<arg>	M-V	None available
Refresh current window		C-L	M-X refresh
Scroll defun to top		C-M-R	M-X scroll defun to top
Scroll down	<arg>	S-Down-arr.	M-X scroll down
Scroll pane to left	<arg>	C-X <	M-X scroll left
Scroll other pane	<arg>	C-M-V	M-X scroll other pane
Scroll right	<arg>	C-X >	M-X scroll right
Scroll Up	<arg>	S-Up-arrow	M-X scroll up
Select the next pane	<arg>	C-X O	M-X select next pane
Select the next window		C-X N or M-Next	M-X select next window

	or S-Next	
Select the previous window	C-X P	M-X select previous window
	or M-Prev	
	or S-Prev	
Select a window by name	C-X W	M-X select window
Set fill column	<arg> C-X F	M-X set fill column
Set fill prefix	<arg> C-X .	M-X set fill prefix
Set goal column	<arg> C-X C-N	M-X set goal column
Set left fill column (in Text mode)	<arg> C-M-Y	M-X set left fill column
Set paragraph indent (in Text mode)	<arg> C-M-P	M-X set paragraph indent
View one pane	C-X 1	M-X view one pane
Window options	n.a.d.	M-X window options

Writing and Editing Text

These commands let you write and edit text (documents, memos, reports). They work in Text mode; most of them also work in Language modes: Lisp, C, Pascal. Some commands are prefixed with <arg> to suggest that the Universal Argument command, **C-U <integer>**, is very useful.

Characters

Delete backward	<arg> Backspace	M-X delete backward hacking tabs
Delete forward	<arg> C-D	M-X delete forward character
	or <arg> Delete char	
Delete horizontal space	<arg> M-\	M-X delete horizontal space
Delete indentation	<arg> M-^	M-X delete indention
Insert next character	<arg> C-Q	M-X insert next character
Move backward	<arg> C-B	M-X move backward character
Move Forward	<arg> C-F	M-X move forward character
Tabbing (in Text Mode)	<arg> M-I	M-X tab to tab stop
	or M-Tab	
Transpose characters	C-T	M-X transpose characters
Simulate backspace	<arg> C-H	none available
Upcase digit	M-'	M-X upcase digit

Words

Delete backward word	<arg> M-Backsp	M-X delete backward word
Delete forward word	<arg> M-D	M-X delete forward word
Lowercase word	<arg> M-L	M-X lowercase word
Move backward	<arg> M-B	M-X move backward word
Move forward	<arg> M-F	M-X move forward word
Transpose words	<arg> M-T	M-X transpose words
Uppercase initial	<arg> M-C	M-X uppercase initial
Uppercase word	<arg> M-U	M-X uppercase word

Lines

Auto fill return	<arg> Return	None available
(provides filling in Auto Fill mode)		
Auto fill space	<arg> Spacebar	None available
(provides filling in Auto Fill mode)		
Back to indentation	M-M	M-X back to indentation
	or C-M-M	
	or C-M-Return	
Center a line	<arg> M-S	M-X center line
Clear the current line	Clear-line	M-X clear line
Delete a line	<arg> C-K	M-X delete line
Delete blank lines	C-X C-O	M-X delete blank lines
Delete matching lines	n.a.d.	M-X delete matching lines
Delete nonmatching lines	n.a.d.	M-X keep lines
Indent a new line	<arg> C-J	M-X indent new line
Move down line extend	<arg> C-N	M-X move down extending
Move down line	<arg> Down-arrow	M-X move down
Move to line end	<arg> C-E	M-X move to end of line
Move to line start	<arg> C-A	M-X move to start of line
Move up line	<arg> C-P	M-X move up
	or <arg> Up-arrow	
Open a line	<arg> C-O	M-X open line
Open line and indent	<arg> M-O	M-X open line indent
	or Insert-line	
Split line	<arg> C-M-O	M-X split line
Tabbing (in Text Mode)	<arg> M-I	M-X tab to tab stop
	or M-Tab	
Transpose lines	<arg> C-X C-T	M-X transpose lines

Sentences

Backward delete sentenc	<arg> C-X Backsp	M-X backward delete sentence
Backward sentence	<arg> M-A	M-X backward sentence
Forward sentence	<arg> M-E	M-X forward sentence
Delete sentence	<arg> M-K	M-X delete sentence
Transpose two sentences	<arg> C-M-T	M-X transpose sentences
(in Text Mode)		

Paragraphs

Back to indentation	C-M-M	M-X back to indentation
	or M-M	
Backward paragraph	<arg> M-[	M-X backward paragraph
Fill paragraph	<arg> M-Q	M-X fill paragraph
Forward paragraph	<arg> M-]	M-X forward paragraph
Delete paragraph	C-X M-K	M-X delete paragraph
Mark paragraph	n.a.d.	M-X mark paragraph
Set fill column	<arg> C-X F	M-X set fill column
Set left fill column	<arg> C-M-Y	M-X set left fill column
(in Text Mode)		
Set paragraph indent	<arg> C-M-P	M-X set paragraph indent

(in Text Mode)			
Transpose paragraph	<arg>	C-X M-T	M-X transpose paragraph

Killing and Yanking Text

Append next kill		C-M-W	M-X append next kill
Backward delete senten.	<arg>	C-X Backsp	M-X backward delete sentence
Delete backward word	<arg>	M-Backsp	M-X delete backward word
Delete forward word	<arg>	M-D	M-X delete forward word
Delete line	<arg>	C-K	M-X delete line
Delete paragraph		C-X M-K	M-X delete paragraph
Delete sentence	<arg>	M-K	M-X delete sentence
Delete region		C-W	M-X delete region
Insert the last kill	<arg>	C-Y	M-X insert from kill ring
Unkill previous	<arg>	M-Y	M-X unkill previous

Assorted Editing Aids

Buffer page left		M-Left-arrow	None available
Buffer page right		M-Right-arro	None available
Center line	<arg>	M-S	M-X center line
Copy region		M-W	M-X copy region
Delete to buffer end		M-Clr dis	M-X delete to buffer end
Expand abbreviation		C-Return	M-X expand abbreviation
	or	S-Return	
Fill comment	<arg>	M-Z	M-X fill comment
Fill region	<arg>	M-G	M-X fill region
Get register		C-X G	M-X get register
Insert date		n.a.d.	M-X insert date
Insert in buffer		n.a.d.	M-X insert in buffer
Move to buffer end		M->	M-X move to buffer end
	or	Shifted Home	
Move to buffer start		M-<	M-X move to buffer start
	or	Home	
Move to pane edge	<arg>	M-R	M-X move to pane edge
Negative argument		C--	M-X negative argument
Put register		C-X X	M-X put register
Set a fill prefix		C-X .	M-X set fill prefix
Set a goal column	<arg>	C-X C-N	M-X set goal column
Set indent to arg.	<arg>	n.a.d.	M-X set indent to argument
Set a left margin	<arg>	C-M-Y	M-X set left margin
Set a right margin	<arg>	C-X F	M-X set right margin
Set undo depth		n.a.d.	M-X set undo depth
Simulate tab	<arg>	C-I	M-X simulate tab
Tab to tab stop	<arg>	M-I	M-X tab to tab stop
Terminate buffer/newline		n.a.d.	M-X terminate buffer with newline
Undo text buffer		C-X U	M-X undo
Universal argument		C-U <int.>	
Upcase digit		M-'	M-X upcase digit
Uppercase initial	<arg>	M-C	M-X uppercase initial

Working With Strings

These commands find, replace, insert, or otherwise deal with strings. Use them in editing text or code.

Count occurrences	n.a.d.	M-X count occurrences
Delete matching lines	n.a.d.	M-X delete matching lines
Delete non-matching lines	n.a.d.	M-X delete non-matching lines
Incremental search	C-S	M-X search
List matching lines	n.a.d.	M-X list matching lines
List nonmatching lines	n.a.d.	M-X list non matching lines
Query replace	M-%	M-X query replace
Replace string	n.a.d.	M-X replace string
Reverse search	C-R	M-X reverse search
Search backward for string	n.a.d.	M-X search backward
Search forward for string	n.a.d.	M-X search forward
Word search	C-M-S	M-X word search

Using Marks in Text and Code

These commands set marks, which when used with the cursor's location and the Copy Region Command, **M-W**, help you effect block moves in editing text or code.

Exchange point and mark	C-X C-X	M-X exchange point and mark
Mark beginning	C- <	M-X mark beginning
Mark defun (in Lisp)	C-M-Backsp	M-X mark defun
Mark end of buffer	C- >	M-X mark end
Mark form (in Lisp)	C-M-@	M-X mark form
Mark a pane for transpose	n.a.d.	M-X mark pane for transpose
Mark paragraph (in text)	M-P	M-X mark paragraph
Mark sentence	C-X M-S	M-X mark sentence
Mark whole buffer	C-X C-H	M-X mark whole buffer
Mark word	M-@ or C-X @	M-X mark word
Set mark	<arg> M-Spacebar or C-Spacebar or C-@	M-X set mark

Writing and Editing Code

Be in a desired language mode (Lisp, C, Pascal, or Fortran) and use the following commands in concert with most of the commands for editing text.

If a command does not appear to work, you might need to load a particular module. The forms that load modules are discussed in the *Installation and Overview* manual, the *Comments:* entry in the *NMODE Command Reference* manual, and the chapters about browsers and editing in the *NMODE User's Guide*.

Comments and Headers

Block comment	n.a.d.	M-X block comment
Fill comment	M-Z	M-X fill comment
Insert comment	M-;	M-X insert comment
Insert date	n.a.d.	M-X insert date
Make header	n.a.d.	M-X make header
Make revisions	n.a.d.	M-X make revisions
Parse header	n.a.d.	M-X parse header

Evaluating Lisp Programs

Evaluate form	C-X C-E	M-X eval form
Execute defun	C-] D	M-X execute defun
Execute from a point	C-]	M-X execute from point
Execute form (Lisp e)	C-] E	M-X execute form

Lisp Break Loops and Errors

Lisp abort	C-] A	M-X lisp abort
Lisp backtrace	C-] B	M-X lisp backtrack
Lisp continue	C-] C	M-X lisp continue
Lisp help	C-] ?	M-X lisp help
Lisp quit	C-] Q	M-X lisp quit

Deleting and Yanking While Programming

Delete backward form	<arg> C-M-DEL	M-X delete backward form
Delete forward form	<arg> C-M-K	M-X delete forward form
Yank last output	C-] Y	M-X yank last output

Examining Code

Browse code	n.a.d.	M-X browse code
Browse occurrences	n.a.d.	M-X browse occurrences
Create code index	n.a.d.	M-X create code index
Create error index	n.a.d.	M-X create error index
Create file search index	n.a.d.	M-X create search index
Find source code item	M-.	M-X find item
Load code index	n.a.d.	M-X load code index
Load search index	n.a.d.	M-X load search index
Register code index	n.a.d.	M-X register code index

Assorted Lisp Programming Aids

Abort edit template	n.a.d.	M-X abort edit template
Backward up list	<arg> C-M-(	M-X backward up list
Delete backward hacking	<arg> Backspace	M-X delete backward hacking tabs
Delete forward form	<arg> C-M-backsp	M-X delete forward form
Down list	<arg> C-M-D	M-X down list
End of defun (in Lisp)	<arg> C-M-E	M-X end of defun
	or C-M-]	
Execution monitor	n.a.d.	M-X execution monitor
Find item (the meta-point)	M-.	M-X find item
Forward up list	<arg> C-M-)	M-X forward up list
Get register	C-X G	M-X get register
Indent new line in Lisp	<arg> C-J	M-X indent new line
Insert closing bracket	)	not available
Insert comment	M-;	M-X insert comment
Insert template	M-O	M-X insert template
Lisp indent region	<arg> C-M-\	M-X lisp indent region
Lisp indent sexpr	C-M-Q	M-X lisp indent sexpr
Lisp tab	C-M-I	M-X lisp tab
Make parens, ()	<arg> M-(	M-X Make Parens
Make a placeholder	n.a.d.	M-X make placeholder
Make revision	n.a.d.	M-X make revision
Make a template	n.a.d.	M-X make template
Move backward defun	<arg> C-M-[	M-X move backward defun
	or C-M-A	
Move backward form (Lis	<arg> C-M-B	M-X move backward form
Move backward list (Lis	<arg> C-M-P	M-X move backward list
Move forward defun	<arg> n.a.d.	M-X move forward defun
Move forward form (Lisp	<arg> C-M-F	M-X move forward form
Move forward list (Lisp	<arg> C-M-N	M-X move forward list
Move over paren	<arg> M-)	M-X move over paren
Move to line end	<arg> C-E	M-X move to end of line
Move to line start	<arg> C-A	M-X move to start of line
Occurrence of string	n.a.d.	M-X count occurences
Open a line	<arg> C-O	M-X open line
Open line and indent	<arg> Insert-line	M-X open line indent
Put register	C-X X	M-X put register
Set fill column	<arg> C-X F	M-X set fill column
Set fill prefix	C-X .	M-X set fill prefix
Set goal column	<arg> C-X C-N	M-X set goal column
Set package	n.a.d.	M-X set package
Show package	n.a.d.	M-X show package
Split line	<arg> C-M-O	M-X split line
Tab	<arg> C-M-I	M-X lisp tab
Tab to tab stop	<arg> M-Tab	M-X tab to tab stop
	or M-I	
Transpose forms	<arg> C-M-T	M-X transpose forms

Assorted C, Fortran, and Pascal Aids

C tab	<arg> C-M-I	M-X c tab
Edit template	n.a.d.	M-X edit template
Delete placeholder	n.a.d.	M-X delete placeholder
Fortran tab	<arg> C-M-I	M-X fortran tab
Get register	C-X G	M-X get register
Insert template header	M-O	M-X insert template header
Lang. insert closing bracket	] or }	None available
(in C Mode)		
Lang. return and indent	<arg> C-J	None available
Make placeholder	n.a.d.	M-X make placeholder
(in Edit Template Mode)		
Match close bracket	M-)	M-X match close bracket
	or C-M-P	
Match close keyword	M-}	M-X match close keyword
	or C-M-B	
Match open bracket	M-(	M-X match open bracket
	or C-M-N	
Match open keyword	M-{	M-X match open keyword
	or C-M-F	
Modify placeholder	n.a.d.	M-X modify placeholder
(in Edit Template Mode)		
Move backward to indent	<arg> M-H	M-X move backward indent
Move forward to indent	<arg> M-I	M-X move forward indent
Move to line end	<arg> C-E	M-X move to end of line
Move to line start	<arg> C-A	M-X move to start of line
Next error (error br.)	C-X M-N	M-X next error
Next placeholder	M-N	M-X next placeholder
Parse header	n.a.d.	M-X parse header
Pascal tab	<arg> C-M-I	M-X pascal tab
Previous error (error br.)	C-X M-P	M-X previous error
Previous placeholder	M-P	M-X previous placeholder
Put register	C-X X	M-X put register
Quit edit template	C-X C-W	M-X quit edit template
	or C-X C-S	
(in C or Pascal Modes)		
Tab-to-tab stop	<arg> M-Tab	M-X tab to tab stop
	or M-I	

Compile Programs

Compile file (Lisp mode)	n.a.d.	M-X compile file
Compile file (C, Pas., For.)	n.a.d.	M-X compile file
Creating error browsers	n.a.d.	M-X create error browser
Next error	C-X M-N	M-X next error
(in C or Pascal error-browser)		
Previous error	C-X M-P	M-X previous error
(in C or Pascal error-browser)		

Regions

These commands manipulate regions established via a mark and the point. Use them in conjunction with editing text and code.

`Copy region`	`M-W`	`M-X copy region`
`Delete region`	`C-W`	`M-X delete region`
`Exchange point and mark`	`C-X C-X`	`M-X exchange point and mark`
`Fill region`	`<arg> M-G`	`M-X fill region`
`Get contents of register`	`C-X G`	`M-X get register`
`Indent region`	`<arg> n.a.d.`	`M-X indent region`
`Indent region in Lisp`	`<arg> C-M-\`	`M-X lisp indent region`
`Insert in buffer`	`n.a.d.`	`M-X insert in buffer`
`Lowercase region`	`C-X C-L`	`M-X lowercase region`
`Put string in register`	`C-X X`	`M-X put register`
`Sort contents of region`	`n.a.d.`	`M-X sort region`
`Transpose regions`	`C-X T`	`M-X transpose region`
`Unkill previous`	`<arg> M-Y`	`M-X unkill previous`
`Uppercase region`	`C-X C-U`	`M-X uppercase region`
`Write region`	`n.a.d.`	`M-X write region`

Buffers

These commands create, kill, browse, and otherwise manipulate buffers. Use them in working with text and code.

`Append to buffer`	`<arg> C-X A`	`M-X append to buffer`
`Buffer not modified`	`M-~`	`M-X buffer not modified`
`Create buffer`	`n.a.d.`	`M-X create buffer`
`Delete to end of buffer`	`n.a.d.`	`M-X delete to buffer end`
`Execute a script`	`n.a.d.`	`M-X execute script buffer`
`Insert buffer`	`n.a.d.`	`M-X insert buffer`
`Insert from kill ring`	`<arg> C-Y`	`M-X insert from kill ring`
`Insert in buffer`	`n.a.d.`	`M-X insert in buffer`
`Kill buffer`	`C-X K`	`M-X kill buffer`
`Kill some buffers`	`n.a.d.`	`M-X Kill some buffers`
`List buffers`	`C-X C-B`	`M-X list buffers`
`Mark whole buffer`	`n.a.d.`	`M-X mark whole buffer`
`Move to buffer end`	`Shift Home`	`M-X move to buffer end`
	`or M->`	
`Move to buffer start`	`Home`	`M-X move to buffer start`
	`or M-<`	
`Print buffer`	`C-X C-P`	`M-X print buffer`
`Rename buffer`	`n.a.d.`	`M-X rename buffer`
`Select buffer (browse)`	`C-X B`	`M-X select buffer`
`Select previous buffer`	`<arg> C-M-L`	`M-X select previous buffer`
`Set undo depth of buffer`	`n.a.d.`	`M-X set undo depth`
`Terminate buffer, newline`	`n.a.d.`	`M-X terminate buffer with newline`

Files (Directories)

These commands let you manipulate files, which can be directories in some cases.

Append to file	n.a.d.	M-X append to file
Change file permission	n.a.d.	M-X change file permission
Compile file	n.a.d.	M-X compile file
Copy file	C-X C-C	M-X copy file
Create a directory	n.a.d.	M-X create directory
Create file	n.a.d.	M-X create file
Dired (directory read)	C-X D	M-X dired
Execute a script	n.a.d.	M-X execute script file
Find an active directory	C-X D	M-X find directory
Find a file	C-X C-F	M-X find file
Insert file in buffer	n.a.d.	M-X insert file
Kill a directory	n.a.d.	M-X kill directory
Kill a file	n.a.d.	M-X kill file
List active directories	n.a.d.	M-X list directories
Move a file	C-X C-R	M-X move file
Prepend to file	n.a.d.	M-X prepend to file
Rename a file	n.a.d.	M-X rename file
Revert to a file	n.a.d.	M-X revert file
Save all files	n.a.d.	M-X save all files
Save file	C-X C-S	M-X save file
Set the visited filename	n.a.d.	M-X set visited filename
Visit a file in a buffer	C-X C-V	M-X visit file
Write buff. cont. to file	C-X C-W	M-X write file
Write window image	n.a.d.	M-X write window image

Browsers

These commands let you manipulate the assorted Nmode browsers.

Browse code (code br)	n.a.d.	M-X browse code
Browse down an item	<arg> Down-arrow or C-N	None available
Browse occurrences (file search browser)	n.a.d.	M-X browse occurrences
Browse up an item	<arg> Up-arrow or C-P	None available
Create code index (code br.)	n.a.d.	M-X create code index
Create directory	n.a.d.	M-X create directory
Create error index (in C, Pas., Fort.)	n.a.d.	M-X create error index
Create search index	n.a.d.	M-X create search index
Find an active directory	C-X D	M-X find directory
List the editing buffers	C-X C-B	M-X list buffers
List active directories	C-X C-D	M-X list directories
List the facilities	C-X R	M-X nmode root or M-X list nmode root
List facility user options	n.a.d.	M-X list options

		or M-X options
Load code index	n.a.d.	M-X load code index
Load error index	n.a.d.	M-X load error index
Load options	n.a.d.	M-X load options
Load search index	n.a.d.	M-X load search index
Register code index	n.a.d.	M-X register code index
Window options	n.a.d.	M-X window options

Miscellaneous Aids

These commands can be used in assorted places, depending on your task and intentions.

Argument digit	C-<digit> or M-<digit>	not available
Begin macro	C-X (	M-X begin macro
Collect space in memory	n.a.d.	M-X make space
Disable the mouse buttons	n.a.d.	M-X disable mouse buttons
Enable the mouse buttons	n.a.d.	M-X enable mouse buttons
End macro	C-X)	M-X end macro
Execute script in a buffer	n.a.d.	M-X execute script buffer
Execute script in a file	n.a.d.	M-X execute script file
Make space (collect garbage)	n.a.d.	M-X make space
Name macro	n.a.d.	M-X name macro
Negative argument	C--	not available
Network connect	n.a.d.	M-X network connect
Network disconnect	n.a.d.	M-X network disconnect
Reset the user bindings	n.a.d.	M-X reset user bindings
Save macro	n.a.d.	M-X save macro
Start the current macro	C-X M	None available
Set package	n.a.d.	M-X set package
Show package	n.a.d.	M-X show package
Start scripting	n.a.d.	M-X start scripting
Stop scripting	n.a.d.	M-X stop scripting
Universal argument	C-U <arg>	None available
User keys mode (toggle)	n.a.d.	M-X user keys mode

General Softkeys

The commands executed by pressing softkeys are discussed in *Appendix C*, which is a liftout card.

General Keyboard

The keys on the keyboard are documented in several places and in several contexts. The *NMODE User's Guide* contains *Appendix D*, which contains tables that show, for each key, the tokens in the device driver for a keyboard. The tables show the normal token and the tokens for various combinations of keys: Shifted, Control, Meta, and Control/Shift.

To see what happens at the user-level when you press a key, see Chapter 2 in the *NMODE User's Guide*. In addition, the *NMODE Command Reference* manual contains a section that explains what happens when you press a key.

Appendix B
Browser Menus

Introduction

Each browser displays a title, a list of items, a mode line, and a menu of commands that appear in the message area (the menu can also be called a command line). Each Browser mode has a fixed menu structure regardless of how you get into the mode; for example, Text-Browsing mode and Help-Help mode are each used in many places, but each has a fixed menu of commands. The menu of commands is available in addition to typical NMODE commands established by the dispatch table for the current browser mode. The functionality that the menu provides is a set of commands specifically intended to let you use a menu-driven routine to manipulate the items in the browser. This appendix describes the commands that occur in menus, presenting them in alphabetical order.

Browser commands have the following characteristics.

- Because characters do not self-insert in a browser mode, you execute a command in a menu by pressing the capital letter in the command's name.
- Each browser has its own menu of commands, although some commands appear in several menus. For example, the commands named `Help`, `Browse`, `Filter`, `Group`, and `Quit` appear in most menus.
- You do not see all available commands as you move among the browsers because many commands in top-level menus do not directly execute a command. Instead, they provide another menu of commands, and some of these commands provide yet another menu. Consequently, part of knowing how to execute a certain command is to know where it is located and know which sequence of commands gets to it.

The commands in menus for browsers work in the following ways:

- Some commands such as `Quit` do something in one step with no intervening action on your part. That is, they perform a task directly.
- Other commands such as `Filter` provide another menu of commands. These commands merely provide access to additional commands.

- Still other commands such as `Write` prompt for input and then perform a task according to that input. Executing Nmode Abort, C-G, during before you complete input cancels (aborts) these commands.

In a browser mode, you can use key sequences, M-X Command names, softkeys, and popup items to execute typical NMODE commands in addition to the commands in menus. A browser mode provides many nonediting NMODE commands and some editing commands that have the same functionality in a browser as they have in an editing buffer, but the context can vary.

Given the assorted browsers, the number of commands, and the nesting of menus, it is difficult to provide one chart that shows all the commands in their respective browsers and menus. The following small chart shows the nesting situation for one command, `Help`. The chart shows the general scheme for nesting of menus and lets you see the structure of the Help facility. You will see some duplication of menus, which illustrates how NMODE can enter certain modes from assorted locations in the environment.

In the chart, commands preceded by an asterisk appear directly in the menu for Help. Commands inside braces, {}, appear under a command under Help. Commands inside brackets, [], appear under the indicated second-menu command. In general, the nesting of menus in NMODE seldom exceeds three levels. For example, to display the current dispatch table for a browser mode, you could execute `Help`, `Key-bindings`, `All-commdands`, and then `Screen`. The first three commands provide menus and the fourth command displays the dispatch table.

Chart of Menus and Commands Provided by Help

```
* Where-am-I
    Enters Help-Help Mode and provides these commands:
      {Leave-help, Quit-help-help}

* Explain
    Enters Documentation or Text-Browsing mode.  Documentation mode
    provides these commands:
      {Help, Browse, Group, Filter, Leave-help, Quit}

      Under Group, you get:
        [Item, Exclude, Matching, Differing, Clear, All, Rev.-all, Quit-group]

      Under Filter, you get:
        [Item, Grouped, Non-grouped, Matching, Keep-matching, Undo, Quit-filter]

* Key-bindings
    {Name, Description, All-commands, Help-help, Leave-help, Quit-key-help}

      Description enters Text-Browsing mode and you get:
        [Explain, Help-help, Leave-help, Quit]

      Under All-commands, you get:
        [Screen, Buffer, file, Printer, Help-help, Leave-help, Quit-list-help]

      Help-help enters Help-Help mode and you get:
        [Leave-help, Quit-help-help]

* Documentation
    Enters the Documentation facility and provides these commands:
      {Help, Browse/execute, Group, Filter, Load, Quit}

* Tutor
    Enters Tutor mode and provides these commands:
      {Help, Browse, Group, Filter, Quit}

* Help-help
    Enters Help-Help mode and provides these commands:
      {Leave-help, Quit-help-help}

* Quit-help
```

The Commands

This section contains an alphabetical list of commands that appear in menus of browsers. The commands are presented alphabetically with information about which commands you need to execute to get to them. The list has the following format:

- You see the command name and an entry inside parentheses. The term `Top`, inside parentheses, indicates that the command appears in a menu at the **top level.** This means that the command is visibly available in a menu when you invoke a browser that has the command. Otherwise, the entry in parentheses shows the sequence of commands you execute to get to the command in a secondary menu. For example, **Description (Help, Key-bindings)** means that you execute `Help`, `Key-bindings`, and then `Description`.
- A description of the command appears under the command name. Some commands such as `Browse` are very common, and in these cases, the description does not mention the specific browsers that provide the command. Other commands such as `File-permission` appear in the menu of a specific browser, and in these cases, the description mentions the related browser or browsers.

The list of commands follows:

Activate-directory (Create)

In the browser for `Directories`, the command activates a directory by reading it into NMODE and browsing into the directory. The specified directory is added to the list of active directories.

Add-files (Top)

In a Code Index, Compilation Error Index, or File Search Index, executing `Add-files` adds the specified file to the list of files in the index. The action taken by the index when a file is added is different for each index.

In a Code Index, adding a file also adds lines extracted from the specified file according to the current extraction rules set by the options for that language.

In a Compilation Error Index, adding a file compiles the file using the currently set compiler options for that language.

In a Search Index, adding a file simply adds an item to the index for the specified file. Execute `Pattern` to specify the pattern to search for in the files listed in the index.

The entered file for any index can by a directory, and you can include the wildcard character, *. For example, entering `mycode/*` adds all files in the directory named `mycode`, while entering `mycode/old*` adds only those files in the directory that begin with the letters `old`.

All (Group)
Groups all items in the current browser's list, making them part of the multiple selection group.

All-commands (Help, Key-bindings)
Executing `All-commands` provides a menu of commands related to getting help.

```
Screen Buffer File Printer Help-help  Leave-help  Quit-list-help
```

The commands named `Screen`, `Buffer`, `File`, and `Printer` output the bindings for the current dispatch table to the specified entity. `Help-help` provides a description of the commands. `Leave-help` exits the Help facility altogether and `Quit-list-help` exits to the menu that contains `All-commands`.

Browse (Top)
Invokes an item for viewing or editing, depending on the item.

Browse-code (Type-specific)
In the browser for a directory, when the current item is a source code file, the command creates and enters a Code Index for the current file. See `Type-specific` to get more information about when this command is available.

Browse/modify (Top)
Invokes or modifies an item in a browser for options, whichever operation suits the item.

Browse/execute (Top)
In the browsers for Additional Facilities and Documentation, the command browses into or executes an item, whichever operation suits the item.

Buffer (Help, Key-bindings, All-commands)
Displays the current dispatch table in a specified buffer.

Buffer-name (Sort)
Sorts buffers alphabetically by name in the browser for Buffers.

Clear (Group)
Ungroups all items in the browser.

Copy (Utility in an active directory)
Copys the highlighted file, or files, to the file or directory name you provide.

Create (Top)
The Create command is provided in several browsers. In all cases, executing `Create` provides a menu of commands that let you create the type of item related to the browser. The command also provides a command named `Other` which gives you alternatives for creating items. Depending on the current browser, commands in the second level menus include: `Buffer`, `Code-index`, `File`, `Directory`, `Directory-browser`, `Error-index` or `User-options`. Except for `Other` and `Quit`, the commands prompt for a name and then create the appropriate item, for example, a buffer or an index.

Current-file (Utilities)
In the browsers for code, search, and error indexes, the command displays the name of the file that was processed to build the current index item (code form, search string, or error).

Description (Help, Key-bindings)
Prompts for entry of a key sequence or M-X Command name and then displays information related to the entry.

Differing (Group)
Prompts for entry of a string and groups all items that do not contain the string in their display lines.

Documentation (Help)
Invokes the top level of online documentation, which is the facility called `Documentation` under `NMODE Root`.

Edit (Type-specific)
In a browser for a directory, executing `Edit` places the current file in a buffer in which you can edit the file. See `Type-specific` to get more information about when this command is available.

Exclude (Group)
Ungroups the current item (removes the >).

Explain (Help)
Prompts for a term for which you want an explanation (in a browser) or defaults to the current word (in an editing buffer). The command then searches for matches, presenting them as a browsable list. On not finding any matches, the command indicates that no matches exist. Avoid entering plural terms.

File (Help, Key-bindings, All-commands)
Prompts for entry of a file name and then writes the current dispatch table to the file.

File-compile (Type-specific)
In the browser for a directory, executing `File-compile` lets you compile a language source file. See `Type-specific` to get more information about when this command is available.

File-name (Sort)
In a browser for buffers or a directory, executing `File-name` sorts items alphabetically by file name.

File-permission (Utility)
In the browser for a directory, the command lets you alter the permissions for the current file in a manner similar to that of HP-UX.

File-revert (Utility)
In the browser for buffers, the command prompts for entry of `Y` or `N` concerning your decision to overwrite the current item's buffer with the existing disc version of the associated file. Entering `Y` lets you revert back to the disc version. Use the command when you have edited a file in such a way that it is no longer acceptable and you need to start over.

Filter (Top)
A pervasive command that provides a menu of commands:

```
Item Grouped Non-grouped Matching Keep-matching Undo Quit
```

The name of the command suggests the criteria for filtering. Filtered items are not destroyed, they are just not displayed. A filter applies during a working session until you Undo it. You can have more than one filter.

Group (Top and several lower-level menus)
A pervasive command that provides a menu of commands, any of which let you include an item in a group.

```
Item  Exclude  Matching  Differing  Clear  All  Reverse-all  Quit
```

See the individual commands to get additional information.

Grouped (Filter)
Hides all grouped items; that is, it hides all items in the multiple selection group.

Help (Top)
Invokes the Help Facility, providing a menu of commands:

```
Where-am-I Explain Key-bindings Documentation Tutor Help-help Quit-help
```

Each command provides help of some type. See the individual commands to get additional information.

Help-help (Help and several other commands)
Displays a description of the commands in the menu for `Help` or `<other command>`. Note that `Help-help` appears at a second or third level and that it always functions in this way; that it, it describes commands in the current menu.

Interpret (Type-specific)
In the browser for a directory, executing `Interpret` lets you evaluate the current Lisp source file. See `Type-specific` to get more information about when this command is available.

Item (Filter)
Hides the current item; that is, it removes the item from the display, but does not destroy or otherwise affect the item.

Item (Group)
Groups an item and displays a > to its left. The > indicates that the item is in the multiple selection group.

Keep-matching (Filter)
Prompts for a string to match, and then hides all items which do not contain the string anywhere in their display lines.

Key-bindings (Help)
Provides a menu of commands that provide assorted types of help relative to bindings between command functions and key sequences or M-X Command names.

```
    Name  Description  All-commands  Help-help  Leave-help  Quit-key-help
```

See the individual commands to get additional information.

Kill (Top or some second levels)
Irreversibly removes a highlighted item. If the kill will remove a file or unsaved buffer, the command prompts for confirmation to kill or not. Use this command with caution and think about what it will do before you press $\boxed{\text{K}}$.

Leave-help (Help, <Some lesser command>)
At whatever level, the command exits the Help Facility and returns to the entity in which you requested help.

Load (Top)
Loads the current item in the browsers for Additional Facilities and Documentation and displays an `L` at the left end of an item, but does not browse the item. You can tell if an item needs to be loaded during a working session because, when you invoke either browser, an `L` appears at the left end of any item that was previously loaded.

Load-code-index (Create)
Loads a code index that was previously saved to a file. The new index is added to the Code Indexes facility.

Load-options-file (Create)
Loads a file of type opt to create a browser for user options.

Load-search-index (Create)
Loads a search index that was previously saved to a file. The new index is added to the File Search Indexes facility.

Load (Type-specific)
Lets you load a compiled file into memory. This command was called Faslin in some other Lisp user environments. See `Type-specific` to get more information about when this command is available.

Matching (Filter)
Prompts for entry of a string and hides (filters) all items that contain the string in their display lines.

Matching (Group)
Prompts for entry of a string and then groups all items that contain the string in their display lines, displaying a > next to the grouped items.

Modified (Sort)
Moves buffers marked by an * to the top of the list. The * indicates the buffers that have been modified.

Move (Utility)
In a browser for a directory, executing `Move` prompts for a directory or file name and then moves the current item to the specified directory or file name. The command removes the item from the original location.

Name (Help, Key-bindings)
Prompts for entry of a key sequence or M-X Command name and returns the name of the command function bound to the entry.

Non-Grouped (Filter)
Hides all items in a list that are not grouped; that is, it hides all items not in the multiple selection group.

Not-modified (Utility)
In the browser for `Buffers`, executing `Not-modified` removes the displayed * from the current buffer item and lets NMODE treat the buffer as if it were not modified.

Options (Top)
This command is available in most browsers. It enters the user options that relate to the current browser. If more than one set of user options relate to the current browser, a list of user-options choices is provided in a secondary menu of commands.

Pattern (Top)
In the browser for file search indexes, the command prompts for entry of a string, which is used as a search pattern, and initiates a search for the string in all included files.

Print (Utility)
In a browser for a directory, executing `Print` prints the highlighted file. The command prints the "in-memory" (buffer) version of a file, if it is present. A prompt asks you to specify the printer spooler.

Print-buffer (Utility)
Prints the contents of the current buffer item. A prompt asks you to specify the printer spooler.

Printer (Help, Key-bindings, All-commands)
Prints the current dispatch table. A prompt asks you to specify the printer spooler.

Quit
Exits the current entity and returns to the previous entity in the current window or pane.

Quit-<command>
Several secondary menus contain a secondary Quit command that quits the named command. Examples include: `Quit-create`, `Quit-sort`, `Quit-help-help`, and `Quit-list-help`.

Re-filter (Filter)
In the table of contents for an online manual, the command filters sub-chapter items untill only the chapter items remain. This browser is initialized with 2 filters active. Re-filter will re-establish one of these filters after a Filter-undo.

Register-code-index (Create)
In the browser for code indexes, the command lets you register a code index that was previously saved to a file. Registering a code index makes the index items available for the Find Item command, **M-.**, but not available for browsing via the Code Indexes facility. A registered code index will take less heap (memory) space than a loaded index, but the Find Item command will run somewhat slower. A registered code index will appear in the Code Indexes facility, and browsing a registered code index will automatically load it.

Rename (Utility)
In the browser for Directories, the command prompts for entry of the new name of the current file and then renames the file.

Rename-buffer (Utility)
In the browser for `Buffers`, the command prompts for entry of a buffer name and then renames the current buffer, using the specified name.

Restore-default (Top)
In the browser for user options, executing `Restore-default` restores the original default value for the current option (system variable).

Reverse (Sort)
Sorts files in the reverse order of a specified criterion. For example, a reverse sort according to `Size` sorts from largest to smallest.

Reverse-all (Group)
Inverts the include-exclude multiple selection group sense of items in a browser.

Run-make (Utilities)
In the browser for search or error indexes, the command invokes the command in HP-UX called *make(1)*, which is executed with arguments determined by the values of the options for `make` in the browser for error indexes. To make full use of this command, you might need to examine the documentation for *make (1)* in the *HP-UX Reference* manual, *Vol. 1B: Section 1 (M through Z)*.

Save (Utility)
In the browser for a directory, the command saves (writes) the current file, if it has been modified, or indicates that no changes need to be written.

Screen (Help, Key-bindings, All-commands)
The command invokes a text-browsing mode and displays the current dispatch table. Text-browsing mode provides the following menu of commands:

```
    Explain  Help-help  Leave-help  Quit
```

that you can execute according to your needs.

Set-filename (Utility)
In the browser for buffers, the command prompts for entry of a file name and then associates the specified file name with the current buffer. Hit Return with no entry to have no file associated with the buffer.

Sort (Top)
In the browsers for buffers or a directory, executing `Sort` provides a menu of commands used as criteria for sorting; for example, `Filename`, `Size`, or `Type`.

Size (Sort)
Sorts files according to file size, smallest to greatest.

Trash (Top or after executing Utility)
Marks the current item for deletion, displaying a `T` at the left of the item, or unmarks an item marked for deletion, removing the `T`. Marked items are not trashed (deleted) until you exit the browser with a Quit or **C-M-L** command.

Tutor (Help)
Invokes the Tutor module, which provides several online tutorials that can help you learn how to use the NMODE environment. If you have not previously loaded the module, the command prompts for a decision to load it.

Type (Sort)
In a browser for a directory, executing `Type` sorts files according to file type, preserving the previous order within a type.

Type-specific (Top)
In a browser for a directory, executing `Type-specific` either: tells you that there are no type-specific commands for the highlighted file; or it provides a menu of commands such as:

```
Browse-code   Edit   Interpret   File-compile   Quit
```

To determine type, the command uses the current file's suffix or standard header. The menu of subcommands varies to accommodate conventions for working with a file of a certain type; for example, you cannot interpret Pascal source code, but such code can be compiled.

Undo (Filter)
Reverses the effect of the last use of the following commands for `Filter`:

```
     Item  Grouped  Non-grouped  Matching  Keep-matching
```

If you have applied several filters, executing `Undo` successively undoes the filters in the order of last applied to first applied. Successive item filters will be undone with a single Undo command.

Update (Utility)
In a browser for a directory, the command re-reads the directory and updates each item. This is necessary after the directory has been modified by an HP-UX shell or Lisp function.

Update (Utility)
In a browser for an index, the command effects any necessary updates in the current item. It does this by processing the file for the current item again and replacing old items with newly processed items. If the current item is not marked (included in a multiple selection group), the command only works on that item. Otherwise, the command processes all marked items in the browser.

Utilities (Top level for browser for code indexes)
Provides a menu of commands that let you work with the current browser.

```
     Update  Write  Kill  Current-file  Quit-util
```

See the individual commands to get additional information.

Utilities (Top level in the browsers for error and search indexes)
Provides a menu of commands that let you work with the current browser.

```
     Update  Run-make  Kill  Write  Current-file  Quit-util
```

See the individual commands to get additional information.

Utility (Top level in browser for a directory)
Provides a menu of commands that let you work with the current browser:

```
    Move  Copy  Rename  Save  Kill Trash  Print  Update  Access  Quit-util
```

See the individual commands to get additional information.

Utility (Top level in browser for buffers)
Provides a menu of commands that let you work with the current browser:

```
 Not-modified Print-buffer Rename-buffer Set-filename File-revert Quit-util
```

See the individual commands to get additional information.

Where-am-I (Help)
The command displays a brief description of the current mode along with some description of the commands in the current menu.

Write (Top level in the browser for buffers)
Writes the current buffer to its associated file if the contents of the buffer have been modified. The command prompts for a file name, offering the associated file as the default.

Write (Utilities)
In a Code Index or Compilation Error index, the command writes (saves) the index data base to a specified file (i.e. it prompts for a file).

Write (Top level of browser for user options)
Writes (saves) the options (with their current values) to a specified file. The command defaults to the current filename in your `*USER-CUSTOMIZE-PREFIX*` directory. See the chapter called *User Options* in the *NMODE User's Guide* for more details.

Write-date (Sort)
Executing `Write-date` sorts files according to the date they were written, earliest to most recent.

Appendix C
Softkeys and Softkey Labels

Introduction

The softkeys, which are the dark gray keys f1 to f8 in the HP 46020A keyboard, provide an alternative for executing some NMODE commands. They distribute evenly about two keys: Menu and a User System combination; the User key is shifted.

Softkeys are available when you run NMODE with HP Windows/9000 software. The softkey labels appear in two lines in a window at the bottom of the display. Softkeys are available on any other primary display (e.g. a terminal), but the softkey labels might be truncated and abbreviated. They are not available on any secondary display. The position of the softkey label relates to the position of the softkey. For example, press f3 to execute the command given by the third softkey label from the left.

If you do **not** load the Nkeys Softkey Interface Package, you get two sets of softkeys; System and User toggle the two sets. If you **do** load this package, some softkeys are mode sensitive; that is, the available commands change as you move among the major and minor modes, and you have more commands. In general, you access commands by pressing System or User and by executing **C-System** or **C-User**. In some cases, you press a softkey to get additional commands. If the softkey labels are not displayed, press Menu, which is bound to the Toggle Menu Keys command and sits between the two sets of softkeys.

This appendix assumes you will load the Nkeys Softkey interface package, and therefore, it shows the softkeys you get in that configuration. Your personal initialization file called .nmoderc has information about loading this package. See the *NMODE Installation and Overview* manual and the *NMODE User's Guide* to get information about this package.

On loading the Nkeys package, you can utilize the following sets of softkeys:

- Pressing User (a shifted key) executes the Setup User Softkeys command, which provides assorted user keys.
- Pressing System executes the Setup Mode Softkeys command, which provides assorted mode-sensitive softkeys; that is, the commands change to fit the mode.

- Executing CTRL User (a shifted key) executes the Display User Keys command, which provides a simple set of keys. They are the same keys you get by pressing User without the Nkeys package.
- Executing CTRL System executes the Display System Keys command, which provides a simple set of keys. They are the same keys you get by pressing System without the Nkeys package.

If you do not want the default softkeys, you can extensively customize the softkey bindings. Information about how to customize softkey bindings is contained in the chapter called *Customization* in the *NMODE User's Guide* and in one section of your initialization file called nmoderc.

Softkey Labels and Commands

This section contains an alphabetical list of commands you can execute by pressing a softkey.

Append to File

Executes the Append To File command, which prompts for entry of a filename and then appends the contents of the current region to the file.

Auto Fill On (Off)

Executes the Auto Fill Mode command. If you have enabled Auto Fill Mode, the label reads **Auto Fill Off**. When Auto Fill Mode is not enabled, the label reads **Auto Fill On**.

Back Trace

When you get into a break loop, the softkey executes the Lisp Backtrace command.

Break: Abort

When you get into a break loop, the softkey executes the Lisp Abort command.

Break: Continue

When you get into a break loop, the softkey executes the Lisp Continue command.

Break: Quit

When you get into a break loop, the the softkey executes the Lisp Quit command.

Buffers
Executes the List Buffers command, which displays a list of current buffers.

Center Line
Executes the Center Line command, which centers the current line.

Complete Filename
In the HP-UX shell buffer in Mode Keys, the softkey executes the Complete Filename command, which attempts to complete any filename you might be entering.

Copy File
Executes the Copy File command, which lets you make a copy of a file.

Create File
Executes the Create File command, which lets you create a new file.

Create Window
Executes the Create Default Window command, which automatically creates a default window and makes it current.

Directories
Executes the List Directories command, which lists the active directories in the facility called `Directories` in NMODE Root.

Exec Defun
In Lisp Mode, executes the Execute Defun command, which evaluates the current function definition within a Lisp source file.

Exec Form
In Lisp Mode, executes the Execute Form command, which evaluates the current program (form) within a Lisp source file.

Execute Line
In an HP-UX shell buffer, executes the HP-UX Execute Line command, which trims off the prompt and executes the entered HP-UX command.

Execute Region
In an HP-UX shell buffer, executes the HP-UX Execute Region command, which submits the current region as an HP-UX command or commands.

Execute to End
In the HP-UX shell buffer, executes the HP-UX Execute to End command, which trims off the current line at the cursor and submits the remainder of the line as an HP-UX command or commands.

Fill Parag.
Executes the Fill Paragraph command, which fills a paragraph.

Find Direct.
Executes the Edit Directory command, which invokes a browser for a specified directory.

Find File
Executes the Find File command, which finds (loads) a file and lets you edit it in an editing buffer.

Format Comment
Executes the Fill Comment command, which fills the segment of a source code file that contains the file header.

Help
Calls the Help Facility.

Horiz. New Pane
Executes the Create Lower Pane command, which creates a new pane and makes it current.

HP-UX Command
Executes the HP-UX Execute By Prompt command, which lets you execute an HP-UX command that you type in the message area.

Indent Form
Executes the Lisp Indent Sexpr command, which indents a form.

Insert File
Executes the Insert File command, which inserts a file into the current buffer, beginning at point.

Jump to Mark
Executes the Exchange Point and Mark command, which exchanges the locations of the current mark and point.

Lisp Abort
The command aborts an arbitrarily deep break loop.

Lisp Command
Executes the Eval Form command, which prompts for entry of a form in the message area and then evaluates the form.

Kill Pane
Executes the Kill Pane command, which kills the current pane in a set of panes.

Kill Window
Executes the Kill Window command, which provides a default and prompts you to press ? to get a list of choices or execute **C-G** to abort.

Kill Region
Executes the Kill Region command, which kills the current region.

Mode
Pressing the softkey for **Mode** provides a set of softkeys:

```
 f1       f2       f3       f4                                  f8

Help     Lisp     Text     Autofill                            Exit
         Mode     Mode      Mode                               Menu
```

The softkeys on the left work as implied. Executing **Exit-Menu** returns to the state in which you executed **Mode**.

Next Keys
Repeatedly pressing the softkey for $\boxed{\text{Next}}$ cycles through several sets of softkeys for $\boxed{\text{f5}}$ through $\boxed{\text{f8}}$.

Next Page
Executes the Display Next Page command, which scrolls the contents of a pane up one page.

Next Pane
Executes the Select Next Pane command, which moves the cursor to the other pane.

Next Parag.
Executes the Forward Paragraph command, which moves the cursor foward to the next paragraph.

Next Window
Executes the Select Next Window command, which makes the next window current.

Next Word
Executes the Move Forward Word command, which moves the cursor forward to the next word.

NMODE Command
Initiates execution of an M-X Command name and displays the prompt:

```
     Enter Command name (? for choices, C-G to Abort):
```

in the message area.

NMode Root
Executes the List Nmode Root command, which displays a list of current facilities in `NMODE Root`.

Previous Keys
This is a shift of `Next Keys` and selects the previous set of softkeys instead of the next set.

Prev Page
Executes the Display Previous Page command, which scrolls the contents of a pane down one page.

Prev Window
Executes the Select Previous Window command, which makes the previous window current.

Query Replace
Executes the Query Replace command, which lets you do a query replace of a string.

Quit
Executes the function bound to **C-M-L**, usually Select Previous Buffer command.

Rename File
Executes the Rename File command, which lets you rename an existing file.

Replace All
Executes the Replace String command, which lets you replace an existing string. The command prompts for the string being replaced and then prompts for entry of the replacement string.

Save File
Executes the Save File command, which automatically writes the contents of a buffer to its associated file, if there is such a file. Prompts for entry of a filename if there is no associated file.

Search
Executes the Incremental Search command, which lets does an incremental search for a string as you enter characters.

Search Back
Executes the Reverse Search command, which works like the Incremental Search command, except that it searches backwards. See *Search.*

Set L Margin
Executes the Set Left Fill Column command, which sets the left fill column to the value determined by the cursor's current location unless you provide a positive argument that is used for the left fill column.

Set Mark
Executes the Set Mark command, which sets a mark at the current location of point.

Set P Indent
Executes the Set Paragraph Indent command, which sets the column for paragraph indentation.

Transp. Chars.
Executes the Transpose Characters command, which transposes the characters either side of point.

Undo
Executes the Text Buffer Undo command, which undoes deletions and insertions in text.

User Options
Executes the Nmode Horizon Options command, which invokes a browser for user options.

Vert. New Pane
Executes the Create Side Pane command, which creates a vertical pane to the right of the current pane and makes the new pane current.

Write File
Executes the Write File command, which saves the current buffer to a specified file.

Yank Last Kill
Executes the Yank Last Kill command, which yanks (reinserts) the contents of the top level of the Kill Ring.

Appendix D
Keyboard Keys

Introduction

Each key on a keyboard in combination with the modifier keys Shift, Control, and Meta produces a unique token. The token as reported by various commands is composed of a base-symbol with optional prefix codes for the modifier keys. These prefix codes are S- for Shift, C- for Control and M- for Meta. There is also a prefix code N- for keys on the number-pad. The prefix codes are allowed in any combination of those listed in the note for the specific key.

These same tokens (without the word Key) are used in customization code to define key-bindings. The letters in the matrix refer to availability notes given at the end of this section.

The Keys and Tokens

This section provides a table of the keys and tokens for the keyboard. Then, the section provides some information and the rules for interpreting the letters in the table.

		- - - - - APPLICABLE RULE - - - - -		
MAIN KEYBOARD KEY LABEL	TOKEN NAME	GRAPHICS WINDOW	S300 ITE	239X TERMINAL
Back space	Back-space Key	Z	U	U
Break	Break Key	B	B	B
Caps	Caps Key	Y	X	X
Clear display	Clear-display Key	Z	U	U
Clear line	Clear-line Key	Z	U	U
Delete char	Delete-char Key	Z	U	U
Delete line	Delete-line Key	Z	U	U
(Down-arrow)	Down-arrow Key	Z,W	S	S
Enter	Enter Key	Z	X	#
f1	f1 Key	Z	U	U
f2	f2 Key	Z	U	U
f3	f3 Key	Z	U	U
f4	f4 Key	Z	U	U
f5	f5 Key	Z	U	U
f6	f6 Key	Z	U	U
f7	f7 Key	Z	U	U
f8	f8 Key	Z	U	U
(Home-arrow)	Home Key	Z	S	S
Insert char	Insert-char Key	Z	U	U
Insert line	Insert-line Key	Z	U	U
(Left-arrow)	Left-arrow Key	Z,W	U	U
Menu	Menu Key	Z	X	L
Next	Next Key	Z	U	U
Prev	Prev Key	Z	U	U
Print	Print Key	C,M	X	X
Reset	Reset Key	C,M	X	X
Return	Return Key	Z	U	U
(Right-arrow)	Right-arrow Key	Z,W	U	U
Select	Select Key	Y	X	X
Stop	Stop Key	Z	X	X
System	System Key	C,M	X	X
Tab	Tab Key	Z	S	S
(Up-arrow)	Up-arrow Key	Z,W	S	S
User	User Key	C,M	X	X
(Space bar)	SPACE	A	A,I	A,I
ESC	ESCAPE	C,M	A,I	A,I
!	!	A	A,I	A,I
"	"	A	A,I	A,I
#	#	A	A,I	A,I
%	%	A	A,I	A,I
&	&	A	A,I	A,I

'	'	A	A,I	A,I
(	(	A	A,I	A,I
)	)	A	A,I	A,I
*	*	A	A,I	A,I
+	+	A	A,I	A,I
,	,	A	A,I	A,I
-	-	A	A,I	A,I
.	.	A	A,I	A,I
/	/	A	A,I	A,I
0	0	A	A,I	A,I
1	1	A	A,I	A,I
2	2	A	A,I	A,I
3	3	A	A,I	A,I
4	4	A	A,I	A,I
5	5	A	A,I	A,I
6	6	A	A,I	A,I
7	7	A	A,I	A,I
8	8	A	A,I	A,I
9	9	A	A,I	A,I
:	:	A	A,I	A,I
;	;	A	A,I	A,I
<	<	A	A,I	A,I
=	=	A	A,I	A,I
>	>	A	A,I	A,I
?	?	A	A,I	A,I
$	$	A	A,I	A,I
@	@	A	A,V	A,I
A	A	A	A	A
B	B	A	A	A
C	C	A	A	A
D	D	A	A	A
E	E	A	A	A
F	F	A	A	A
G	G	A	A	A
H	H	A	A,Q	A,Q
I	I	A	A,T	A,T
J	J	A	A	A
K	K	A	A	A
L	L	A	A	A
M	M	A	A,R	A,R
N	N	A	A	A
O	O	A	A	A
P	P	A	A	A
Q	Q	A	A	A
R	R	A	A	A
S	S	A	A	A
T	T	A	A	A
U	U	A	A	A
V	V	A	A	A
W	W	A	A	A

X	X	A	A	A
Y	Y	A	A	A
Z	Z	A	A	A
[	[	A	A,E	A,E
\	\	A	A	A
]	]	A	A	A
^	^	A	A	A
_	_	A	A	A
`	`	A	A,V	A,@
{	{	A	A,E	A,E
|	|	A	A,\	A,\
}	}	A	A,]	A,]
~	~	A	A,^	A
DEL	DEL	C,M	A,_	A
Shift	S-	D	D	D
CTRL	C-	F	F	F
Extend char (left)	M-	O	J	X
Extend char (right)		P	J	X

		- - - - - APPLICABLE RULE - - - - -		
NUMERIC KEYPAD KEY LABEL	TOKEN NAME	GRAPHICS WINDOW	S300 ITE	239X TERMINAL
Enter	N-Enter Key	Z,N	X	#
(f9)	N-f9 Key	Z,N	X	X
(f10)	N-f10 Key	Z,N	X	X
(f11)	N-f11 Key	Z,N	X	X
(f12)	N-f12 Key	Z,N	X	X
Tab	N-Tab Key	Z,N	S	S
*	N-*	A,*	U	U
+	N-+	A,*	U	U
,	N-,	A,*	U	U
-	N--	A,*	U	U
.	N-.	A,*	U	U
/	N-/	A,*	U	U
0	N-0	A,*	U	U
1	N-1	A,*	U	U
2	N-2	A,*	U	U
3	N-3	A,*	U	U
4	N-4	A,*	U	U
5	N-5	A,*	U	U
6	N-6	A,*	U	U
7	N-7	A,*	U	U
8	N-8	A,*	U	U
9	N-9	A,*	U	U

Note the following information:

a. Parentheses around a KEY LABEL indicate that the key has no label, or that it has an arrow as a label. The arrows could perhaps be formatted graphically into the real manual. In fact, graphic representations of key caps could be used instead of the key labels, if there was some way to call attention to which of the two labels on multi-labeled keys was being accessed.

b. The HP 46020A keyboard is split into the normal ("main keyboard") and the "numeric pad" sections. Within each section, the keys are listed in the following order. First, the special (ie, "non-ascii") keys appear, in alphabetic order by key label. The ascii keys follow, in numeric order.

c. The USASCII HP 46020A keyboard is the basis for the key labels.

d. There are two labels on some key caps. The key label field in this table refers to the individual label (part of the key cap), which requires the SHIFT key to be in a known position.

e. The M- modifier key is only available with the GRAPHICS WINDOW.

The following items are the applicability rules for the above table. The letter below denotes the rule for the corresponding letter in the table.

a. The alphanumeric key for the S- prefix is not shown. A Shift of this key produces an appropriate character, according to the caps-lock state. Both the shifted and unshifted characters will be an appropriate USASCII character or national character. The key has C- and M- variants unless noted otherwise. (eg under non GRAPHICS WINDOW interfaces the M- variant is never available).

b. The key produces a break signal; no key code is available.

c. The C- modifier is available.

d. This key produces an S- modifier when pressed with some other key. This key works with the caps-lock state and alphanumeric keys to produce a different character instead of an S- modifier.

e. The C- modifier produces the ESCAPE character

f. This key produces a C- modifier when pressed with some other key.

i. The C- modifier is ignored.

j. In 8 bit mode this key, when held down, causes alphanumeric keys to be interpreted as if from a ROMAN 8 keyboard.

l. The C- variant is bound to no-op-command, other variants are not available.

m. The M- modifier is available.

n. The N- modifier is used with this key.

o. When held down, this key produces a M- modifier for any other key pressed at the same time. For the Katakana keyboard this key ALSO causes a state to be entered where the alpha keys are interpreted from the USASCII keyboard. (See note P)

p. When held down, this key causes any alpha key pressed at the same time to be interpreted as if from a ROMAN 8 keyboard. For the Katakana keyboard this key causes entry into a state where the alpha keys are interpreted as Katakana keys. (See note O)

When held down, this key produces a C- modifier for any other key pressed at the same time.

q. The C- modifier produces the Back-space Key.

r. The C- modifier produces the Return Key.

s. S- modifier available. (other modifiers may be ignored)

t. C- modifier produces the Tab Key.

u. S- and C- modifiers are ignored.

v. C- variants are not available (may be used by terminal emulator)

w. C- variants are bound to the no-op-command to avoid confilict with terminal emulator usage.

x. The key is not avaialble to NMODE. (may be used by terminal emulator)

y. The key is available to NMODE but all variants are bound to the no-op-command to avoid confilict with terminal emulator usage.

z. The S-, C-, and M- modifiers are available.

There are some special cases.

\	The C- modifier produces C-\
]	The C- modifier produces C-]
^	The C- modifier produces C-^
_	The C- modifier produces C-_
@	The C- modifier produces C-@

* The N- modifier is available except in combination with the Shift key.

# The action of this key is determined by the terminal emulator (unless programmed to some sequence recognized by NMODE it is best not touched)

MANUAL COMMENT CARD

NMODE User's Guide
for HP 9000 Series 300 Computers
Manual Reorder No. 98678-90020

Name: ______________________________

Company: ______________________________

Address: ______________________________

____

Phone No: ______________

Please note the latest printing date from the Printing History (page ii) of this manual and any applicable update(s); so we know which material you are commenting on ______________.

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO 37 LOVELAND, COLORADO

POSTAGE WILL BE PAID BY ADDRESSEE

Hewlett-Packard Company
Attn: Customer Documentation
3404 East Harmony Road
Fort Collins, Colorado 80525

HEWLETT
PACKARD

HP Part Number
98678-90020

Microfiche No. 98678-99020
Printed in U.S.A. 5/86

98678-90601
For Internal Use Only